(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 August 2000 (17.08.2000)

PCT

(10) International Publication Number
WO 00/48171 A1

(51) International Patent Classification⁷: G10L 21/02

(21) International Application Number: PCT/US00/03372

(22) International Filing Date: 9 February 2000 (09.02.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/119,279	9 February 1999 (09.02.1999)	US
09/499,985	8 February 2000 (08.02.2000)	US

(71) Applicant: AT & T CORP. [US/US]; 32 Avenue of the Americas, New York, NY 10013-2412 (US).

(72) Inventors: COX, Richard, Vandervoort; 155 Sagamore Drive, New Providence, NJ 07974 (US). MARTIN, Ranier; Pfalzgrafenstrasse 71, D-52072 Aachen (DE).

(74) Agents: CONOVER, Michele, L. et al.; AT & T Corp., P.O. Box 4110, Middletown, NJ 07748-4110 (US).

(81) Designated States *(national)*: BR, CA, JP, KR.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

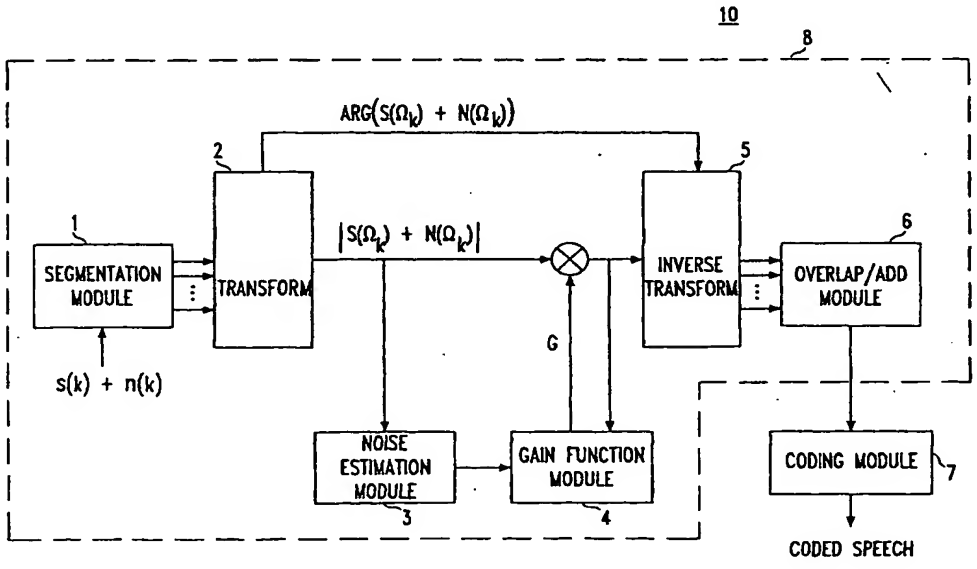
Published:
— *with international search report*

(48) Date of publication of this corrected version:
20 September 2001

(15) Information about Corrections:
see PCT Gazette No. 38/2001 of 20 September 2001, Section II
Previous Correction:
see PCT Gazette No. 14/2001 of 5 April 2001, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: SPEECH ENHANCEMENT WITH GAIN LIMITATIONS BASED ON SPEECH ACTIVITY



(57) Abstract: An apparatus and method for data processing that improves estimation of spectral parameters of speech data and reduces algorithmic delay in a data coding operation. Estimation of spectral parameters is improved by adaptively adjusting a gain function used to enhance data based on whether the data contains information speech and noise or noise only. Delay is reduced by extracting coding parameters using incompletely processed data.

WO 00/48171 A1

Speech Enhancement With Gain Limitations Based On Speech Activity

Cross-Reference to Related Applications

This application claims the benefit of the filing date of U.S. Provisional Application No. 60/119,279, filed February 9, 1999, and is incorporated herein by reference.

Field of the Invention

This invention relates to enhancement processing for speech coding (*i.e.*, speech compression) systems, including low bit-rate speech coding systems such as MELP.

Background of the Invention

Low bit-rate speech coders, such as parametric speech coders, have improved significantly in recent years. However, low-bit rate coders still suffer from a lack of robustness in harsh acoustic environments. For example, artifacts introduced by low bit-rate parametric coders in medium and low signal-to-noise ratio (SNR) conditions can affect intelligibility of coded speech.

Tests show that significant improvements in coded speech can be made when a low bit-rate speech coder is combined with a speech enhancement preprocessor. Such enhancement preprocessors typically have three main components: a spectral analysis/synthesis system (usually realized by a windowed fast Fourier transform/inverse fast Fourier transform (FFT/IFFT), a noise estimation process, and a spectral gain computation. The noise estimation process typically involves some type of voice activity detection or spectral minimum tracking technique. The computed spectral gain is applied only to the Fourier magnitudes of each data frame (*i.e.*, segment) of a speech signal. An example of a speech enhancement preprocessor is provided in Y. Ephraim et al., "Speech Enhancement Using a Minimum Mean-Square Error Log-Spectral

Amplitude Estimator," IEEE Trans. Acoustics, Speech and Signal Processing, Vol. 33, pp. 443-445, April 1985, which is hereby incorporated by reference in its entirety. As is conventional, the spectral gain comprises individual gain values to be applied to the individual subbands output by the FFT process.

A speech signal may be viewed as representing periods of articulated speech (that is, periods of "speech activity") and speech pauses. A pause in articulated speech results in the speech signal representing background noise only, while a period of speech activity results in the speech signal representing both articulated speech and background noise. Enhancement preprocessors function to apply a relatively low gain during periods of speech pauses (since it is desirable to attenuate noise) and a higher gain during periods of speech (to lessen the attenuation of what has been articulated). However, switching from a low to a high gain value to reflect, for example, the onset of speech activity after a pause, and *vice-versa*, can result in structured "musical" (or "tonal") noise artifacts which are displeasing to the listener. In addition, enhancement preprocessors themselves can introduce degradations in speech intelligibility as can speech coders used with such preprocessors.

To address the problem of structured musical noise, some enhancement preprocessors uniformly limit the gain values applied to all data frames of the speech signal. Typically, this is done by limiting an "*a priori*" signal to noise ratio (SNR) which is a functional input to the computation of the gain. This limitation on gain prevents the gain applied in certain data frames (such as data frames corresponding to speech pauses) from dropping too low and contributing to significant changes in gain between data frames (and thus, structured musical noise). However, this limitation on gain does not adequately ameliorate the intelligibility problem introduced by the enhancement preprocessor or the speech coder.

Summary of the Invention

The present invention overcomes the problems of the prior art to both limit structured musical noise and increase speech intelligibility. In the context of an enhancement preprocessor, an illustrative embodiment of the invention makes a determination of whether the speech signal to be processed represents articulated speech or a speech pause and forms a unique gain to be applied to the speech signal. The gain is unique in this context because the lowest value the gain may assume (*i.e.*, its lower limit) is determined based on whether the speech signal is known to represent articulated speech or not. In accordance with this embodiment, the lower limit of the gain during periods of speech pause is constrained to be higher than the lower limit of the gain during periods of speech activity.

In the context of this embodiment, the gain that is applied to a data frame of the speech signal is adaptively limited based on limited *a priori* SNR values. These *a priori* SNR values are limited based on (a) whether articulated speech is detected in the frame and (b) a long term SNR for frames representing speech. A voice activity detector can be used to distinguish between frames containing articulated speech and frames that contain speech pauses. Thus, the lower limit of *a priori* SNR values may be computed to be a first value for a frame representing articulated speech and a different second value, greater than the first value, for a frame representing a speech pause. Smoothing of the lower limit of the *a priori* SNR values is performed using a first order recursive system to provide smooth transitions between active speech and speech pause segments of the signal.

An embodiment of the invention may also provide for reduced delay of coded speech data that can be caused by the enhancement preprocessor in combination with a speech coder. Delay of the enhancement preprocessor and coder can be reduced by having the coder operate, at least partially, on incomplete data samples to extract at least some coder parameters. The total delay imposed by the preprocessor and coder is usually equal to the sum of the delay of the coder and the length of overlapping portions of frames in the enhancement preprocessor. However, the invention takes advantage of the fact that some coders store "look-ahead" data samples in an input buffer and use these samples to extract coder parameters. The look-ahead samples typically have less influence on the quality of coded speech than other samples in the input buffer. Thus, in some cases, the coder does not need to wait for a fully processed, *i.e.*, complete, data frame from the preprocessor, but instead can extract coder parameters from incomplete data samples in the input buffer. By operating on incomplete data samples, delay of the enhancement preprocessor and coder can be reduced without significantly affecting the quality of the coded data.

For example, delay in a speech preprocessor and speech coder combination can be reduced by multiplying an input frame by an analysis window and enhancing the frame in the enhancement preprocessor. After the frame is enhanced, the left half of the frame is multiplied by a synthesis window and the right half is multiplied by an inverse analysis window. The synthesis window can be different from the analysis window, but preferably is the same as the analysis window. The frame is then added to the speech coder input buffer, and coder parameters are extracted using the frame. After coder parameters are extracted, the right half of the frame in the speech coder input buffer is multiplied by the analysis and the synthesis window, and the frame is shifted in the input buffer before the next frame is input. The analysis windows, and synthesis window used to process the frame in the coder input buffer can be the same as the analysis and synthesis windows used in the enhancement preprocessor, or can be slightly different, *e.g.*, the square root of the analysis window used in the preprocessor. Thus, the delay imposed by the preprocessor can be reduced to a very small level, *e.g.*, 1-2 milliseconds.

These and other aspects of the invention will be appreciated and/or obvious in view of the following description of the invention.

Brief Description of the Drawings

The invention is described in connection with the following drawings where reference numerals indicate like elements and wherein:

Figure 1 is a schematic block diagram of an illustrative embodiment of the invention.

Figure 2 is a flowchart of steps for a method of processing speech and other signals in accordance with the embodiment of Figure 1.

Figure 3 is a flowchart of steps for a method for enhancing speech signals in accordance with the embodiment of Figure 1.

Figure 4 is a flowchart of steps for a method of adaptively adjusting an *a priori* SNR value in accordance with the embodiment of Figure 1.

Figure 5 is a flowchart of the steps for a method of applying a limit to the *a priori* signal to noise ratio for use in a gain computation.

Detailed Description

A. Introduction to Illustrative Embodiments

As is conventional in the speech coding art, the illustrative embodiment of the present invention is presented as comprising individual functional blocks (or "modules"). The functions these blocks represent may be provided through the use of either shared or dedicated hardware, including, but not limited to, hardware capable of executing software. For example, the functions of blocks 1-5 presented in Figure 1 may be provided by a single shared processor. (Use of the term "processor" should not be construed to refer exclusively to hardware capable of executing software.)

Illustrative embodiments may be realized with digital signal processor (DSP) or general purpose personal computer (PC) hardware, available from any of a number of manufacturers, read-only memory (ROM) for storing software performing the operations discussed below, and random access memory (RAM) for storing DSP/PC results. Very large scale integration (VLSI) hardware embodiments, as well as custom VLSI circuitry in combination with a general purpose DSP/PC circuit, may also be provided.

Illustrative software for performing the functions presented in Figure 1 is provided in the Software Appendix hereto.

B. The Illustrative Embodiment

Figure 1 presents a schematic block diagram of an illustrative embodiment 8 of the invention. As shown in Figure 1, the illustrative embodiment processes various signals representing speech information. These signals include a speech signal (which includes a pure speech component, s(k), and a background noise component, n(k)), data frames thereof, spectral magnitudes, spectral phases, and coded speech. In this example, the speech signal is enhanced by a speech enhancement preprocessor 8 and then coded by a coder 7. The coder 7 in this illustrative embodiment is a 2400 bps MIL Standard MELP coder, such as that described in A. McCree et al., "A 2.4 KBIT/S MELP Coder Candidate for the New U.S. Federal Standard," Proc., IEEE Intl. Conf. Acoustics, Speech, Signal Processing (ICASSP), pp. 200-203, 1996, which is hereby incorporated by reference in its entirety. Figures 2, 3, 4, and 5 present flow diagrams of the processes carried out by the modules presented in Figure 1.

1. *The Segmentation Module*

The speech signal, s(k) + n(k), is input into a segmentation module 1. The segmentation module 1 segments the speech signal into frames of 256 samples of speech and noise data (*see* step 100 of Figure 2; the size of the data frame can be any desired size, such as the illustrative 256 samples), and applies an analysis window to the frames prior to transforming the frames into the frequency domain (*see* step 200 of Figure 2). As is well known, applying the analysis window to the frame affects the spectral representation of the speech signal.

The analysis window is tapered at both ends to reduce cross talk between subbands in the frame. Providing a long taper for the analysis window significantly reduces cross talk, but can result in increased delay of the preprocessor and coder combination 10. The delay inherent in the preprocessing and coding operations can be minimized when the frame advance (or a multiple thereof) of the enhancement preprocessor 8 matches the frame advance of the coder 7. However, as the shift between later synthesized frames in the enhancement preprocessor 8 increases from the typical half-overlap (*e.g.*, 128 samples) to the typical frame shift of the coder 7 (*e.g.*, 180 samples), transitions between adjacent frames of the enhanced speech signal $\hat{s}(k)$ become less smooth. These discontinuities arise because the analysis window attenuates the input signal most at the edges of each frame and the estimation errors within each frame tend to spread out evenly over the entire frame. This leads to larger relative errors at the frame boundaries, and the resulting discontinuities, which are most notable for low SNR conditions, can lead to pitch estimation errors, for example.

Discontinuities may be greatly reduced if both an analysis and synthesis windows are used in the enhancement preprocessor 8. For example, the square root of the Tukey window

$$w(i) = \begin{cases} \sqrt{0.5(1-\cos(\pi i / M_0))} & \text{for } 1 \le i \le M_0 \\ \sqrt{0.5(1-\cos(\pi(M-i)/M_0))} & \text{for } M - M_0 \le i \le M \\ 1 & \text{otherwise} \end{cases} \qquad (1)$$

gives good performance when used as both an analysis and a synthesis window. M is the frame size in samples and M_0 is the length of overlapping sections of adjacent synthesis frames.

Windowed frames of speech data are next enhanced. This enhancement step is referenced generally as step 300 of Figure 2 and more particularly as the sequence of steps in Figures 3, 4, and 5.

2. *The Transform Module*

The windowed frames of the speech signal are output to a transform module 2, which applies a conventional fast Fourier transform (FFT) to the frame (*see* step 310 of Figure 3). Spectral magnitudes output by the transform module 2 are used by a noise estimation module 3 to estimate the level of noise in the frame.

3. *The Noise Estimation Module*

The noise estimation module 3 receives as input the spectral magnitudes output by the transform module 2 and generates a noise estimate for output to the gain function module 4 (*see* step 320 of Figure 3). The noise estimate includes conventionally computed *a priori* and *a posteriori* SNRs. The noise estimation module 3 can be realized with any conventional noise estimation technique, and may be realized in accordance with the noise estimation technique presented in the above-referenced U.S. Provisional Application No. 60/119,279, filed February 9, 1999.

4. *The Gain Function Module*

To prevent musical distortions and avoid distorting the overall spectral shape of speech sounds (and thus avoid disturbing the estimation of spectral parameters), the lower limit of the gain, G, must be set to a first value for frames which represent background noise only (a speech pause) and to a second lower value for frames which represent active speech. These limits and the gain are determined illustratively as follows.

4.1 *Limiting the a priori SNR*

The gain function, G, determined by module 4 is a function of an *a priori* SNR value ξ_k and an *a posteriori* SNR value γ_k (referenced above). The *a priori* SNR value ξ_k is adaptively limited by the gain function module 4 based on whether the current frame contains speech and noise or noise only, and based on an estimated long term SNR for the speech data. If the current frame contains noise only (*see* step 331 of Figure 4), a preliminary lower limit $\xi_{min1}(\lambda) = 0.12$ is preferably set for the *a priori* SNR value ξ_k (*see* step 332 of Figure 4). If the current frame contains speech and noise (*i.e.*, active speech), the preliminary lower limit $\xi_{min1}(\lambda)$ is set to

$$\xi_{min1}(\lambda) = 0.12 \exp(-5)(0.5 + SNR_{LT}(\lambda))^{0.65} \qquad (3)$$

where SNR_{LT} is the long term SNR for the speech data, and λ is the frame index for the current frame (*see* step 333 of Figure 4). However, ξ_{min1} is limited to be no greater than 0.25 (*see* steps 334 and 335 of Figure 4). The long term SNR_{LT} is determined by generating the ratio of the average power of the speech signal to the average power of the noise over multiple frames and subtracting 1 from the generated ratio. Preferably, the speech signal and the noise are averaged over a number of frames that represent 1-2 seconds of the signal. If the SNR_{LT} is less than 0, the SNR_{LT} is set equal to 0.

The actual lower limit for the *a priori* SNR is determined by a first order recursive filter:

$$\xi_{min}(\lambda) = 0.9\xi_{min}(\lambda-1) + 0.1\xi_{min1}(\lambda) \qquad (4)$$

This filter provides for a smooth transition between the preliminary values for speech frames and noise only frames (*see* step 336 of Figure 4). The smoothed lower limit $\xi_{min}(\lambda)$ is then used as the lower limit for the *a priori* SNR value $\xi_k(\lambda)$ in the gain computation discussed below.

4.2 *Determining the Gain with a Limited a priori SNR*

As is known in the art, gain, G, used in speech enhancement preprocessors is a function of the *a priori* signal to noise ratio, ξ, and the *a posteriori* SNR value, γ. That is, $G_k = f(\xi_k(\lambda), \gamma_k(\lambda))$, where λ is the frame index and k is the subband index. In accordance with an embodiment of this invention, the lower limit of the *a priori* SNR, $\xi_{min}(\lambda)$, is applied to the *a priori* SNR (which is determined by noise estimation module 3) the as follows:

$$\xi_k(\lambda) = \xi_k(\lambda) \text{ if } \xi_k(\lambda) > \xi_{min}(\lambda)$$

$$\xi_k(\lambda) = \xi_{min}(\lambda) \text{ if } \xi_k(\lambda) \leq \xi_{min}(\lambda)$$

(*see* steps 510 and 520 of Figure 5).

Based on the *a posteriori* SNR estimation generated by the noise estimation module 3 and the limited *a priori* SNR discussed above, the gain function module 4 determines a gain function, G (*see* step 530 Figure 5). A suitable gain function for use in realizing this embodiment is a conventional Minimum Mean Square Error Log Spectral Amplitude estimator (MMSE LSA), such as the one described in Y. Ephraim et al., "Speech Enhancement Using a Minimum Mean-Square Error Log-Spectral Amplitude Estimator," IEEE Trans. Acoustics, Speech and Signal Processing, Vol. 33, pp. 443-445, April 1985, which is hereby incorporated by reference as if set forth fully herein. Further improvement can be obtained by using a multiplicatively modified MMSE LSA estimator, such as that described in D. Malah, et al., "Tracking Speech Presence Uncertainty to Improve Speech Enhancement in Non-Stationary Noise Environments," Proc. ICASSP, 1999, to account for the probability of speech presence. This reference is incorporated by reference as if set forth fully herein.

5. *Applying the Gain Function*

The gain, G, is applied to the noisy spectral magnitudes of the data frame output by the transform module 2. This is done in conventional fashion by multiplying the noisy spectral magnitudes by the gain, as shown in Figure 1 (*see* step 340 of Figure 3).

6. *The Inverse Transform Module*

A conventional inverse FFT is applied to the enhanced spectral amplitudes by the inverse transform module 5, which outputs a frame of enhanced speech to an overlap/add module 6 (*see* step 350 of Figure 3).

7. *Overlap Add Module; Delay Reduction*

The overlap/add module 6 synthesizes the output of the inverse transform module 5 and outputs the enhanced speech signal $\hat{s}(k)$ to the coder 7. Preferably, the overlap/add module 6 reduces the delay imposed by the enhancement preprocessor 8 by multiplying the left "half" (*e.g.*, the less current 180 samples) in the frame by a synthesis window and the right half (*e.g.*, the more current 76 samples) in the frame by an inverse analysis window (*see* step 400 of Figure 2). The synthesis window can be different from the analysis window, but preferably is the same as the analysis window (in addition, these windows are preferably the same as the analysis window referenced in step 200 of Figure 2). The sample sizes of the left and right "halves" of the frame will vary based on the amount of data shift that occurs in the coder 7 input buffer as discussed below (*see* the discussion relating to step 800, below). In this case, the data in the coder 7 input buffer is shifted by 180 samples. Thus, the left half of the frame includes 180 samples. Since the analysis/synthesis windows have a high attenuation at the frame edges, multiplying the frame by the inverse analysis filter will greatly amplify estimation errors at the frame boundaries. Thus, a small delay of 2-3 ms is preferably provided so that the inverse analysis filter is not multiplied by the last 16-24 samples of the frame.

Once the frame is adjusted by the synthesis and inverse analysis windows, the frame is then provided to the input buffer (not shown) of the coder 7 (*see* step 500 of Figure 2). The left portion of the current frame is overlapped with the right half of the previous frame that is already loaded into the input buffer. The right portion of the current frame, however, is not overlapped with any frame or portion of a frame in the input buffer. The coder 7 then uses the data in the input buffer, including the newly input frame and the incomplete right half data, to extract coding parameters (*see* step 600 of Figure 2). For example, a conventional MELP coder extracts 10 linear prediction coefficients, 2 gain factors, 1 pitch value, 5 bandpass voicing strength values, 10 Fourier magnitudes, and an aperiodic flag from data in its input buffer. However, any desired information can be extracted from the frame. Since the MELP coder 7 does not use the latest 60 samples in the input buffer for the Linear Predictive Coefficient (LPC) analysis or computation of the first gain factor, any enhancement errors in these samples have a low impact on the overall performance of the coder 7.

After the coder 7 extracts coding parameters, the right half of the last input frame (*e.g.*, the more current 76 samples) are multiplied by the analysis and synthesis windows (*see* step 700 of Figure 2). These analysis and synthesis windows are preferably the same as those referenced in step 200, above (however, they could be different, such as the square-root of the analysis window of step 200).

Next, the data in the input buffer is shifted in preparation for input of the next frame, *e.g.*, the data is shifted by 180 samples (see step 800 of Figure 2). As discussed above, the analysis and synthesis windows can be the same as the analysis window used in the enhancement preprocessor 8, or can be different from the analysis window, *e.g.*, the square root of the analysis window. By shifting the final part of overlap/add operations into the coder 7 input buffer, the delay of the enhancement preprocessor 8/coder 7 combination can be reduced to 2-3 milliseconds without sacrificing spectral resolution or cross talk reduction in the enhancement preprocessor 8.

C. Discussion

While the invention has been described in conjunction with specific embodiments thereof, it is evident that many alternatives, modifications and variations will be apparent to those skilled in the art. Accordingly, the preferred embodiments of the invention as set forth herein are intended to be illustrative, not limiting. Various changes may be made without departing from the spirit and scope of the invention.

For example, while the illustrative embodiment of the present invention is presented as operating in conjunction with a conventional MELP speech coder, other speech coders can be used in conjunction with the invention.

The illustrative embodiment of the present invention employs an FFT and IFFT, however, other transforms may be used in realizing the present invention, such as a discrete Fourier transform (DFT) and inverse DFT.

While the noise estimation technique in the referenced provisional patent application is suitable for the noise estimation module 3, other algorithms may also be used such as those based on voice activity detection or a spectral minimum tracking approach, such as described in D. Malah et al., "Tracking Speech Presence Uncertainty to Improve Speech Enhancement in Non-Stationary Noise Environments," Proc. IEEE Intl. Conf. Acoustics, Speech, Signal Processing (ICASSP), 1999; or R. Martin, "Spectral Subtraction Based on Minimum Statistics, " Proc. European Signal Processing Conference, vol. 1, 1994, which are hereby incorporated by reference in their entirety.

Although the preliminary lower limit $\xi_{min1}(\lambda) = 0.12$ is preferably set for the *a priori* SNR value ξ_k when a frame represents a speech pause (background noise only), this preliminary lower limit ξ_{min1} could be set to other values as well.

The process of limiting the *a priori* SNR is but one possible mechanism for limiting the gain values applied to the noisy spectral magnitudes. However, other methods of limiting the gain values could be employed. It is advantageous that the lower limit of the gain values for frames representing speech activity be less than the lower limit of the gain values for frames representing background noise only. However, this advantage could be achieved other ways, such as, for example, the direct limitation of gain values (rather than the limitation of a functional antecedent of the gain, like *a priori* SNR).

Although frames output from the inverse transform module 5 of the enhancement preprocessor 8 are preferably processed as described above to reduce the delay imposed by the enhancement preprocessor 8, this delay reduction processing is not required to accomplish enhancement. Thus, the enhancement preprocessor 8 could operate to enhance the speech signal through gain limitation as illustratively discussed above (for example, by adaptively limiting the *a priori* SNR value ξ_k). Likewise, delay reduction as illustratively discussed above does not require use of the gain limitation process.

Delay in other types of data processing operations can be reduced by applying a first process on a first portion of a data frame, *i.e.*, any group of data, and applying a second process to a second portion of the data frame. The first and second processes could involve any desired processing, including enhancement processing. Next, the frame is combined with other data so that the first portion of the frame is combined with other data. Information, such as coding parameters, are extracted from the frame including the combined data. After the information is extracted, a third process is applied to the second portion of the frame in preparation for combination with data in another frame.

SOFTWARE APPENDIX

1

melp.c.

```
/*

2.4 kbps MELP Federal Standard speech coder

version 1.2

Copyright (c) 1996, Texas Instruments, Inc.

Vishu Viswanathan
Personal Systems Laboratory
Corporate R&D
Texas Instruments
P.O. Box 655303, M/S 8374
Dallas, TX 75265

This Mixed Excitation Linear Prediction (MELP) speech coding algorithm
   luding the C source code software, the pre-existing MELP software and any
     comments thereto, is delivered to the Government in accordance with the
r.   rement of Contract MDA904-94-C-6101.  It is delivered with Government
Purpose License Rights in the field of secure voice communications only.  No
other use is authorized or granted by Texas Instruments Incorporated. The
Government Purpose license rights shall be effective until 30 September 2001;
thereafter, the Government purpose license rights will expire and the
Government shall have unlimited rights in the software.  The restrictions
governing use of the software marked with this legend are set forth in the
definition of "Government Purpose License Rights" in paragraph (a)(14) of the
clause at 252.227-7013 of the contract listed above.   This legend, together
with the indications of the portions of this software which are subject to
Government purpose license rights shall be included on any reproduction hereof
which includes any part of the portions subject to such limitations.

*/

/*                                                                */
/*  melp.c: Mixed Excitation LPC speech coder                     */
/*                                                                */

/*  compiler include files  */
#include <stdio.h>
#include "melp.h"
#include "spbstd.h"
#    ude "mat.h"
#    ude <fstream.h>


/*  ----------------------------------------------------------
       Functions added by atno
    ----------------------------------------------------------  */

void melp_enc( float speech_in[], unsigned int chan_bit[],
              struct melp_param *par, struct melp_param* new_par)
{
        unsigned int chbuf[CHSIZE];
        int i;
        int maxloop;

        par->chptr = chbuf;
        par->chbit = 0;

        melp_ana(speech_in, par, new_par);

        maxloop = par->chptr-chbuf;
        for (i=0; i<maxloop; i++)
                chan_bit[i] = chbuf[i];
```

```
} /* melp_enc */

void melp_enc( float spe_in[], float spe_in_lpc[], float spe_in_pitch[], unsigned int
chan_bit[],
                struct melp_param *par, struct melp_param* new_par)
{
        unsigned int chbuf[CHSIZE];
        int i;
        int maxloop;

        par->chptr = chbuf;
        par->chbit = 0;

        melp_ana(spe_in, spe_in_lpc,spe_in_pitch,par,new_par);

        maxloop = par->chptr-chbuf;
        for (i=0; i<maxloop; i++)
                chan_bit[i] = chbuf[i];

} /* melp_enc */


void melp_dec( unsigned int chan_bit[], float speech_out[],
                struct melp_param *par)
{
        unsigned int chbuf[CHSIZE];
        int i;

        for (i=0; i<CHSIZE; i++)
                chbuf[i] = chan_bit[i];

        par->chptr = chbuf;
        par->chbit = 0;

        melp_syn(par, speech_out);

} /* melp_dec */
```

2/05/99
6:16:06

melp_ana.c.

```
kbps MELP Federal Standard speech coder

ion 1.2

right (c) 1996, Texas Instruments, Inc.

u Viswanathan
onal Systems Laboratory
orate R&D
s Instruments
 Box 655303, M/S 8374
as, TX 75265

 Mixed Excitation Linear Prediction (MELP) speech coding algorithm
udi   the C source code software, the pre-existing MELP software and any
n     .s thereto, is delivered to the Government in accordance with the
-l.   nt of Contract MDA904-94-C-6101.  It is delivered with Government
use License Rights in the field of secure voice communications only.  No
r use is authorized or granted by Texas Instruments Incorporated. The
nment Purpose license rights shall be effective until 30 September 2001,
eafter, the Government purpose license rights will expire and the
rnment shall have unlimited rights in the software.  The restrictions
rning use of the software marked with this legend are set forth in the
nition of "Government Purpose License Rights" in paragraph (a)(14) of the
se at 252.227-7013) of the contract listed above.   This legend, together
 the indications of the portions of this software which are subject to
rnment purpose license rights shall be included on any reproduction hereof
h includes any part of the portions subject to such limitations.



Name: melp_ana.c
Description: MELP analysis
Inputs:
  speech[] - input speech signal
Outputs:
  *par - MELP parameter structure
Returns: void


 mp   : include files */

lude <stdio.h>
lude <math.h>
lude "melp.h"
lude "spbstd.h"
lude "lpc.h"
lude "mat.h"
lude "vq.h"
lude "fs.h"
lude "pit.h"
lude <fstream.h>

mpiler constants */

ine BEGIN 0
ine END 1
ine BWFACT 0.994
ine PDECAY 0.95
ine PEAK_THRESH 1.34
ine PEAK_THR2 1.6
ine SILENCE_DB 30.0
```

```
#define MAX_ORD LPF_ORD
#define FRAME_BEG (PITCHMAX-(FRAME/2))      // 70
#define FRAME_END (FRAME_BEG+FRAME)         // 250
#define PITCH_BEG (FRAME_END-PITCHMAX)      // 90
#define PITCH_FR ((2*PITCHMAX)+1)           // 321
#define DELAY 24  // this shifts data in the input buffer to the right
#define WINCOMP 70
#define IN_BEG (PITCH_BEG+PITCH_FR-FRAME+DELAY)
#define SIG_LENGTH (LPF_ORD+PITCH_FR)


/* external memory references */

extern Float melp_win_cof[LPC_FRAME];
extern Float melp_lpf_num[LPF_ORD+1];
extern Float melp_lpf_den[LPF_ORD+1];
extern Float melp_msvq_cb[];
extern Float melp_fsvq_cb[];
extern int melp_fsvq_weighted;
extern int framemode;                // RM, 07/20/98
extern int autocorrmode;             // RM, 08/04/98
extern int filtermode;
extern int lpcmode;
extern int pitchmode;
extern int readmode;
extern Float r1[LPC_ORD+1];

/* memory definitions */

static Float sigbuf[SIG_LENGTH];
static Float speech[IN_BEG+FRAME+DELAY];
static Float speech_lpc[IN_BEG+FRAME+DELAY];
static Float speech_pitch[IN_BEG+FRAME+DELAY];
static Float dcdel[DC_ORD];
static Float dcdel_lpc[DC_ORD];
static Float dcdel_pitch[DC_ORD];
static Float lpfsp_del[LPF_ORD];
static Float pitch_avg;
static Float fpitch[2];
static struct melp_msvq_param vq_par;  /* MSVQ parameters */
static struct melp_msvq_param fs_vq_par;  /* Fourier series VQ parameters */
static Float w_fs[NUM_HARM];

static Float sqrttukeystart[76] = {
    2.0666901e-02,4.1324974e-02,6.1965395e-02,8.2579345e-02,1.0315002e-01,
    1.2369263e-01,1.4417440e-01,1.6459459e-01,1.8494447e-01,2.0521534e-01,
    2.2539856e-01,2.4548549e-01,2.6546756e-01,2.8533622e-01,3.0508301e-01,
    3.2469947e-01,3.4417723e-01,3.6350797e-01,3.8268343e-01,4.0169542e-01,
    4.2053583e-01,4.3919659e-01,4.5766974e-01,4.7594739e-01,4.9402174e-01,
    5.1188505e-01,5.2952970e-01,5.4694816e-01,5.6413298e-01,5.8107682e-01,
    5.9777244e-01,6.1421271e-01,6.3039062e-01,6.4629924e-01,6.6193178e-01,
    6.7728157e-01,6.9234205e-01,7.0710678e-01,7.2156946e-01,7.3572391e-01,
    7.4956408e-01,7.6308407e-01,7.7627809e-01,7.8914051e-01,8.0166584e-01,
    8.1384872e-01,8.2568395e-01,8.3716648e-01,8.4829140e-01,8.5905395e-01,
    8.6944955e-01,8.7947375e-01,8.8912227e-01,8.9839098e-01,9.0727593e-01,
    9.1577333e-01,9.2387953e-01,9.3159109e-01,9.3890470e-01,9.4581724e-01,
    9.5232576e-01,9.5842748e-01,9.6411979e-01,9.6940027e-01,9.7426664e-01,
    9.7871685e-01,9.8274897e-01,9.8636130e-01,9.8955229e-01,9.9232058e-01,
    9.9466498e-01,9.9658449e-01,9.9807830e-01,9.9914576e-01,9.9978642e-01,
    1.0000000e+00};

static Float sqrttukeyend[76] = {
    9.9978642e-01,9.9914576e-01,9.9807830e-01,9.9658449e-01,9.9466498e-01,
    9.9232058e-01,9.8955229e-01,9.8636130e-01,9.8274897e-01,9.7871685e-01,
    9.7426664e-01,9.6940027e-01,9.6411979e-01,9.5842748e-01,9.5232576e-01,
```

melp_ana.c.

```
.4581724e-01,9.3890470e-01,9.3159109e-01,9.2387953e-01,9.1577333e-01,
.0727593e-01,8.9839098e-01,8.8912227e-01,8.7947375e-01,8.6944955e-01,
5905395e-01,8.4829140e-01,8.3716648e-01,8.2568395e-01,8.1384872e-01,
0166584e-01,7.8914051e-01,7.7627809e-01,7.6308407e-01,7.4956408e-01,
3572391e-01,7.2156946e-01,7.0710678e-01,6.9234205e-01,6.7728157e-01,
6193178e-01,6.4629924e-01,6.3039062e-01,6.1421271e-01,5.9777244e-01,
8107682e-01,5.6413298e-01,5.4694816e-01,5.2953970e-01,5.1188505e-01,
9402174e-01,4.7594739e-01,4.5766974e-01,4.3919659e-01,4.2053583e-01,
0169542e-01,3.8268343e-01,3.6350797e-01,3.4417723e-01,3.2469947e-01,
0508301e-01,2.8533622e-01,2.6546756e-01,2.4548549e-01,2.2539856e-01,
0521534e-01,1.8494447e-01,1.6459459e-01,1.4417440e-01,1.2369263e-01,
0315802e-01,8.2579345e-02,6.1965395e-02,4.1324974e-02,2.0666901e-02,
0000000e+00};

ic Float sqrtsqrttukeyend[76] = {
9978642e-01,9.9914576e-01,9.9807830e-01,9.9658449e-01,9.9466490e-01,
9232058e-01,9.8955239e-01,9.8636130e-01,9.8274897e-01,9.7871685e-01,
7426664e-01,9.6940027e-01,9.6411979e-01,9.5842748e-01,9.5232576e-01,
4581724e-01,9.3890470e-01,9.3159109e-01,9.2387953e-01,9.1577333e-01,
07   3e-01,8.9839098e-01,8.8912227e-01,8.7947375e-01,8.6944955e-01,
'     5e-01,8.4829140e-01,8.3716648e-01,8.2568395e-01,8.1384872e-01,
6  .584e-01,7.8914051e-01,7.7627809e-01,7.6308407e-01,7.4956408e-01,
3572391e-01,7.2156946e-01,7.0710678e-01,6.9234205e-01,6.7728157e-01,
6193178e-01,6.4629924e-01,6.3039062e-01,6.1421271e-01,5.9777244e-01,
8107682e-01,5.6413298e-01,5.4694816e-01,5.2953970e-01,5.1188505e-01,
9402174e-01,4.7594739e-01,4.5766974e-01,4.3919659e-01,4.2053583e-01,
0169542e-01,3.8268343e-01,3.6350797e-01,3.4417723e-01,3.2469947e-01,
0508301e-01,2.8533622e-01,2.6546756e-01,2.4548549e-01,2.2539856e-01,
0521534e-01,1.8494447e-01,1.6459459e-01,1.4417440e-01,1.2369263e-01,
0315802e-01,8.2579345e-02,6.1965395e-02,4.1324974e-02,2.0666901e-02,
0000000e+00};


melp_ana(Float sp_in[],Float sp_in_lpc[],Float sp_in_pitch[],struct melp_param* p
struct melp_param* new_par)

int i;
int begin;
float sub_pitch;
float temp,pcorr,bpthresh;
float r[LPC_ORD+1],refc[LPC_ORD+1],lpc[LPC_ORD+1];
float weights[LPC_ORD];

/*     or (i =0; i< 76; i++)
        tsqrttukeyend[i] = sqrt(sqrttukeyend[i]);
*/
if (framemode == 0) {        // RM, 07/20/98
  if (filtermode == 0) {
     melp_v_equ(&speech[IN_BEG],sp_in,FRAME);
     melp_v_equ(&speech_lpc[IN_BEG],sp_in_lpc,FRAME);
     melp_v_equ(&speech_pitch[IN_BEG],sp_in_pitch,FRAME);
  }
  else {
     melp_dc_rmv(sp_in,&speech[IN_BEG],dcdel,FRAME);
     melp_dc_rmv(sp_in_lpc,&speech_lpc[IN_BEG],dcdel_lpc,FRAME);
     melp_dc_rmv(sp_in_pitch,&speech_pitch[IN_BEG],dcdel_pitch,FRAME);

  };
'
else    // OVERLAP_ADD- vector overlap add operation. RM 07/20/98
  {
  melp_v_cmult(sp_in,sqrttukeystart,76);
  melp_v_add(&speech[IN_BEG-INFRAME+FRAME],sp_in,INFRAME-FRAME);
  melp_v_equ(&speech[IN_BEG],&sp_in[INFRAME-FRAME],FRAME);

     if (lpcmode == 1) {
          melp_v_cmult(sp_in_lpc,sqrttukeystart,76);
          melp_v_add(&speech_lpc[IN_BEG-INFRAME+FRAME],sp_in_lpc,INFRAME-FRAME);
          melp_v_equ(&speech_lpc[IN_BEG],&sp_in_lpc[INFRAME-FRAME],FRAME);
     }
     else {
          melp_v_add(&speech_lpc[IN_BEG-INFRAME+FRAME],sp_in,INFRAME-FRAME);
          melp_v_equ(&speech_lpc[IN_BEG],&sp_in[INFRAME-FRAME],FRAME);
     };

     if (pitchmode ==  1) {
          melp_v_cmult(sp_in_pitch,sqrttukeystart,76);
          melp_v_add(&speech_pitch[IN_BEG-INFRAME+FRAME],sp_in_pitch,INFRAME-FRAME);
          melp_v_equ(&speech_pitch[IN_BEG],&sp_in_pitch[INFRAME-FRAME],FRAME);
     }
     else {
          melp_v_add(&speech_pitch[IN_BEG-INFRAME+FRAME],sp_in,INFRAME-FRAME);
          melp_v_equ(&speech_pitch[IN_BEG],&sp_in[INFRAME-FRAME],FRAME);
     };

   };

   /* multiply end of buffer with inverse analysis window */
   if (frammode == 1)    {        // RM, 07/20/98
       melp_v_cdiv(&speech[IN_BEG+FRAME-INFRAME+FRAME],sqrtsqrttukeyend,WINCOMP);
       melp_v_cdiv(&speech_lpc[IN_BEG+FRAME-INFRAME+FRAME],sqrtsqrttukeyend,WINCOMP);
       melp_v_cdiv(&speech_pitch[IN_BEG+FRAME-INFRAME+FRAME],sqrtsqrttukeyend,WINCOMP
*)
   };

   /* ofstream track2("track2",ofstream::app);
       if (!track2) { cerr << "Cannot open track2 output file!" << endl;
            exit(1);
            }
         for (int icnt =0 ; icnt < 10;icnt++)
     cout << speech[icnt] << '|' << speech_lpc[icnt] << '|' <<  speech_pitch[icnt] <<
'\n';
     for (int icnt = IN_BEG-300; icnt < IN_BEG+FRAME;icnt++)
     track2 << speech[icnt] << '\n'; */

   /* Copy input speech to pitch window and lowpass filter */
   melp_v_equ(&sigbuf[LPF_ORD],&speech_pitch[PITCH_BEG],PITCH_FR);
   melp_v_equ(sigbuf,lpfsp_del,LPF_ORD);
   melp_polflt(&sigbuf[LPF_ORD], melp_lpf_den, &sigbuf[LPF_ORD],
       LPF_ORD, PITCH_FR);
   melp_v_equ(lpfsp_del,&sigbuf[FRAME],LPF_ORD);
   melp_zerflt(&sigbuf[LPF_ORD], melp_lpf_num, &sigbuf[LPF_ORD],
       LPF_ORD, PITCH_FR);

   /* Perform global pitch search at frame end on lowpass speech signal */
   /* Note: avoid short pitches due to formant tracking */
   fpitch[END] = melp_find_pitch(&sigbuf[LPF_ORD+(PITCH_FR/2)],&temp,
                             (2*PITCHMIN),PITCHMAX,PITCHMAX);

   /* Perform bandpass voicing analysis for end of frame */
   melp_bpvc_ana(&speech_pitch[FRAME_END], fpitch, &par->bpvc[0], &sub_pitch);

   /* Force jitter if lowest band voicing strength is weak */
   if (par->bpvc[0] < VJIT)
       par->jitter = MAX_JITTER;
   else
       par->jitter = 0.0;

   /* Calculate LPC for end of frame */
```

2/05/99
:16:06

melp_ana.c.

3

```
if (autocorrmode == 0) {
   melp_window(&speech_lpc[(FRAME_END-(LPC_FRAME/2))],melp_win_cof,sigbuf,LPC_FRAM

   melp_autocorr(sigbuf,r,LPC_ORD,LPC_FRAME);
   lpc[0] = 1.0;
   melp_lpc_schur(r,lpc,refc,LPC_ORD);
}
else {
   lpc[0] = 1.0;
   melp_lpc_schur(r1,lpc,refc,LPC_ORD);
};

/*
cout << r[0] << " " << r[1] << " " << r[2] << " " << r[3] << " " << r[4] << " " <<
] << " " << r[6] << " " << r[7] << " " << r[8] << " " << r[9] << " " << r[10] << "
< endl;
/'     : << r1[0] << " " << r1[1] << " " << r1[2] << " " << r1[3] << " " << r1[4] <
.    .1[5] << endl; //" " << r1[6] << r1[7] << r1[8] << r1[9] << r1[10] << endl;
// cout << endl;
*/
melp_lpc_bw_expand(lpc,lpc,BWFACT,LPC_ORD);

/* Calculate LPC residual */
melp_zerflt(&speech_lpc[PITCH_BEG],lpc,&sigbuf[LPF_ORD],LPC_ORD,PITCH_FR);

/* Check peakiness of residual signal */
begin = (LPF_ORD+(PITCHMAX/2));
temp = melp_peakiness(&sigbuf[begin],PITCHMAX);

/* Peakiness: force lowest band to be voiced  */
if (temp > PEAK_THRESH) {
    per->bpvc[0] = 1.0;
}

/* Extreme peakiness: force second and third bands to be voiced */
if (temp > PEAK_THR2) {
    per->bpvc[1] = 1.0;
    per->bpvc[2] = 1.0;
}

/* C 'culate overall frame pitch using lowpass filtered residual */
pe.   tch = melp_pitch_ana(&speech_pitch[FRAME_END], &sigbuf[LPF_ORD+PITCHMAX],
                     sub_pitch,pitch_avg,&pcorr);

bpthresh = BPTHRESH;

/* Calculate gain of input speech for each gain subframe */
for (i = 0; i < NUM_GAINFR; i++) {
    if (per->bpvc[0] > bpthresh) {

        /* voiced mode: pitch synchronous window length */
        temp = sub_pitch;
        per->gain[i] = melp_gain_ana(&speech[FRAME_BEG+(i+1)*GAINFR],
                                temp,MIN_GAINFR,2*PITCHMAX);
    }
    else {
        temp = 1.33*GAINFR - 0.5;
        per->gain[i] = melp_gain_ana(&speech[FRAME_BEG+(i+1)*GAINFR],
                                temp,0,2*PITCHMAX);



 /* Update average pitch value */
 if (per->gain[NUM_GAINFR-1] > SILENCE_DB)
   temp = pcorr;
```

```
else
  temp = 0.0;
pitch_avg = melp_p_avg_update(per->pitch, temp, VMIN);

/* Calculate Line Spectral Frequencies */

melp_lpc_pred2lsp(lpc,per->lsf,LPC_ORD);

if (readmode ==1 && new_par != NULL) {
   // modify the unquantized parameters

   per->lsf[1] = new_par->lsf[1];
   per->lsf[2] = new_par->lsf[2];
   per->lsf[3] = new_par->lsf[3];
   per->lsf[4] = new_par->lsf[4];
   per->lsf[5] = new_par->lsf[5];
   per->lsf[6] = new_par->lsf[6];
   per->lsf[7] = new_par->lsf[7];
   per->lsf[8] = new_par->lsf[8];
   per->lsf[9] = new_par->lsf[9];
   per->lsf[10] =new_par->lsf[10];

 };

 /* Force minimum LSF bandwidth (separation) */
 melp_lpc_clamp(per->lsf,BWMIN,LPC_ORD);

 /* Quantize MELP parameters to 2400 bps and generate bitstream */

 /* Quantize LSF's with MSVQ */
 melp_vq_lspw(weights, &per->lsf[1], lpc, LPC_ORD);
 melp_msvq_enc(&per->lsf[1], weights, &per->lsf[1], vq_par);
 per->msvq_index = vq_par.indices;

 /* Force minimum LSF bandwidth (separation) */
 melp_lpc_clamp(per->lsf,BWMIN,LPC_ORD);

 /* Quantize logarithmic pitch period */
 /* Reserve all zero-code for completely unvoiced */
 per->pitch = log10(per->pitch);
 melp_quant_u(&per->pitch,&per->pitch_index,PIT_QLO,PIT_QUP,PIT_QLEV);
 per->pitch = pow(10.0,per->pitch);

 /* Quantize gain terms with uniform log quantizer   */
 melp_q_gain(per->gain, per->gain_index,GN_QLO,GN_QUP,GN_QLEV);

 /* Quantize jitter and bandpass voicing */
 melp_quant_u(&per->jitter,&per->jit_index,0.0,MAX_JITTER,2);
 per->uv_flag = melp_q_bpvc(&per->bpvc[0],&per->bpvc_index,bpthresh,
                     NUM_BANDS);

 /*  Calculate Fourier coefficients of residual signal from quantized LPC */
 melp_fill(per->fs_mag,1.0,NUM_HARM);
 if (per->bpvc[0] > bpthresh) {
     melp_lpc_lsp2pred(per->lsf,lpc,LPC_ORD);
     melp_zerflt(&speech_lpc[(FRAME_END-(LPC_FRAME/2))],lpc,sigbuf,
             LPC_ORD,LPC_FRAME);
     melp_window(sigbuf,melp_win_cof,sigbuf,LPC_FRAME);
     melp_find_harm(sigbuf, per->fs_mag, per->pitch, NUM_HARM, LPC_FRAME);
 }

 /* quantize Fourier coefficients */
 /* pre-weight vector, then use Euclidean distance */
 melp_window(&per->fs_mag[0],w_fs,&per->fs_mag[0],NUM_HARM);
 melp_fsvq_enc(&per->fs_mag[0], &per->fs_mag[0], fs_vq_par);
```

melp_ana.c.

```
/* Set MELP indeces to point to same array */
par->fsvq_index = fs_vq_par.indices;

/* Update MSVQ information */
par->msvq_stages = vq_par.num_stages;
par->msvq_bits = vq_par.num_bits;

/* Write channel bitstream */
melp_chn_write(par);

/* Update delay buffers for next frame */
if (framemode == 1)   {        // RM, 07/20/98
    melp_v_cmult(&speech[IN_BEG+FRAME-INFRAME+FRAME],sqrttukeyend,76);
    melp_v_cmult(&speech_lpc[IN_BEG+FRAME-INFRAME+FRAME],sqrttukeyend,76);
    melp_v_cmult(&speech_pitch[IN_BEG+FRAME-INFRAME+FRAME],sqrttukeyend,76);
};

/   date delay buffers for next frame */
if ((framemode == 1))   {        // RM, 07/20/98
    melp_v_cmult(&speech[IN_BEG+FRAME-INFRAME+FRAME],sqrtsqrttukeyend,WINCOMP);
    melp_v_cmult(&speech_lpc[IN_BEG+FRAME-INFRAME+FRAME],sqrtsqrttukeyend,WINCOMP)

    melp_v_cmult(&speech_pitch[IN_BEG+FRAME-INFRAME+FRAME],sqrtsqrttukeyend,WINCOM

};

melp_v_equ(&speech[0],&speech[FRAME],IN_BEG);
melp_v_equ(&speech_lpc[0],&speech_lpc[FRAME],IN_BEG);
melp_v_equ(&speech_pitch[0],&speech_pitch[FRAME],IN_BEG);
fpitch[BEGIN] = fpitch[END];


 melp_ana(Float sp_in[],struct melp_param *par , struct melp_param* new_par)


int i;
int begin;
Float sub_pitch;
Float temp,pcorr,bpthresh;
Float r[LPC_ORD+1],refc[LPC_ORD+1],lpc[LPC_ORD+1];
Fir   weights(LPC_ORD);

/*    ove DC from input speech */
// melp_dc_rmv(sp_in,&speech[IN_BEG],dcdel,FRAME);



if ((framemode == 0)           // RM, 07/20/98
  melp_v_equ(&speech[IN_BEG],sp_in,FRAME);
// melp_dc_rmv(sp_in,&speech[IN_BEG],dcdel,FRAME);

else    // OVERLAP_ADD- vector overlap add operation. RM 07/20/98
  {
  melp_v_cmult(sp_in,sqrttukeystart,76);
  melp_v_add(&speech[IN_BEG-INFRAME+FRAME],sp_in,INFRAME-FRAME);
  melp_v_equ(&speech[IN_BEG],&sp_in[INFRAME-FRAME],FRAME);

};

/* ofstream track2("track2",ofstream::app);
    if (!track2) { cerr << "Cannot open track2 output file!" << endl;
          exit(1);
          }
 for (int icnt = IN_BEG-300; icnt < IN_BEG+FRAME;icnt++)
 track2 << speech[icnt] << '\n'; */

/* Copy input speech to pitch window and lowpass filter */
melp_v_equ(&sigbuf[LPF_ORD],&speech[PITCH_BEG],PITCH_FR);
melp_v_equ(sigbuf,lpfsp_del,LPF_ORD);
melp_polflt(&sigbuf[LPF_ORD], melp_lpf_den, &sigbuf[LPF_ORD],
    LPF_ORD, PITCH_FR);
melp_v_equ(lpfsp_del,&sigbuf[FRAME],LPF_ORD);
melp_zerflt(&sigbuf[LPF_ORD], melp_lpf_num, &sigbuf[LPF_ORD],
    LPF_ORD, PITCH_FR);

/* Perform global pitch search at frame end on lowpass speech signal */
/* Note: avoid short pitches due to formant tracking */
fpitch[END] = melp_find_pitch(&sigbuf[LPF_ORD+(PITCH_FR/2)],&temp,
                       (2*PITCHMIN),PITCHMAX,PITCHMAX);

/* Perform bandpass voicing analysis for end of frame */
melp_bpvc_ana(&speech[FRAME_END], fpitch, &par->bpvc[0], &sub_pitch);

/* Force jitter if lowest band voicing strength is weak */
if (par->bpvc[0] < VJIT)
    par->jitter = MAX_JITTER;
else
    par->jitter = 0.0;

/* Calculate LPC for end of frame */

if (autocorrmode == 0) {
   melp_window(&speech[(FRAME_END-(LPC_FRAME/2))],melp_win_cof,sigbuf,LPC_FRAME);
   melp_autocorr(sigbuf,r,LPC_ORD,LPC_FRAME);
   lpc[0] = 1.0;
   melp_lpc_schur(r,lpc,refc,LPC_ORD);
}
else {
   lpc[0] = 1.0;
   melp_lpc_schur(r1,lpc,refc,LPC_ORD);
};

// cout << r[0] << " " << r[1] << " " << r[2] << " " << r[3] << " " << r[4] << " "
<< r[5] << endl; //" " << r[6] << r[7] << r[8] << r[9] << r[10] << endl;
// cout << r1[0] << " " << r1[1] << " " << r1[2] << " " << r1[3] << " " << r1[4] <
< " " << r1[5] << endl; //" " << r1[6] << r1[7] << r1[8] << r1[9] << r1[10] << endl;
// cout << endl;

melp_lpc_bw_expand(lpc,lpc,BWFACT,LPC_ORD);

/* Calculate LPC residual */
melp_zerflt(&speech[PITCH_BEG],lpc,&sigbuf[LPF_ORD],LPC_ORD,PITCH_FR);

/* Check peakiness of residual signal */
begin = (LPF_ORD+(PITCHMAX/2));
temp = melp_peakiness(&sigbuf[begin],PITCHMAX);

/* Peakiness: force lowest band to be voiced  */
if (temp > PEAK_THRESH) {
    par->bpvc[0] = 1.0;
}

/* Extreme peakiness: force second and third bands to be voiced */
if (temp > PEAK_THR2) {
    par->bpvc[1] = 1.0;
    par->bpvc[2] = 1.0;
}

/* Calculate overall frame pitch using lowpass filtered residual */
```

/05/99
:16:06

melp_ana.c.

```
ar->pitch = melp_pitch_ana(&speech[FRAME_END], &sigbuf[LPF_ORD+PITCHMAX],
                           sub_pitch,pitch_avg,&pcorr);
uthresh = BPTHRESH;

* Calculate gain of input speech for each gain subframe */
or (i = 0; i < NUM_GAINFR; i++) {
   if (par->bpvc[0] > bpthresh) {

       /* voiced mode: pitch synchronous window length */
       temp = sub_pitch;
       par->gain[i] = melp_gain_ana(&speech[FRAME_BEG+(i+1)*GAINFR],
                                    temp,MIN_GAINFR,2*PITCHMAX);
   }
   else {
       temp = 1.33*GAINFR - 0.5;
       par->gain[i] = melp_gain_ana(&speech[FRAME_BEG+(i+1)*GAINFR],
                                    temp,0,2*PITCHMAX);



/* Update average pitch value */
f (par->gain[NUM_GAINFR-1] > SILENCE_DB)
  temp = pcorr;
lse
  temp = 0.0;
itch_avg = melp_p_avg_update(par->pitch, temp, VMIN);

/* Calculate Line Spectral Frequencies */
melp_lpc_pred2lsp(lpc,par->lsf,LPC_ORD);

if (readmode ==1 && new_par != NULL) {
   // modify the unquantised parameters
   par->lsf[1] = new_par->lsf[1];
   par->lsf[2] = new_par->lsf[2];
   par->lsf[3] = new_par->lsf[3];
   par->lsf[4] = new_par->lsf[4];
   par->lsf[5] = new_par->lsf[5];
   par->lsf[6] = new_par->lsf[6];
   par->lsf[7] = new_par->lsf[7];
   par->lsf[8] = new_par->lsf[8];
   par->lsf[9] = new_par->lsf[9];
   par->lsf[10] =new_par->lsf[10];
};

/* Force minimum LSF bandwidth (separation) */
melp_lpc_clamp(par->lsf,BWMIN,LPC_ORD);

/* Quantise MELP parameters to 2400 bps and generate bitstream */

/* Quantise LSF's with MSVQ */
melp_vq_lspw(weights, &par->lsf[1], lpc, LPC_ORD);
melp_msvq_enc(&par->lsf[1], weights, &par->lsf[1], vq_par);
par->msvq_index = vq_par.indices;

/* Force minimum LSF bandwidth (separation) */
melp_lpc_clamp(par->lsf,BWMIN,LPC_ORD);

/* Quantise logarithmic pitch period */
/* Reserve all zero code for completely unvoiced */
par->pitch = log10(par->pitch);
melp_quant_u(&par->pitch,&par->pitch_index,PIT_QLO,PIT_QUP,PIT_QLEV);
par->pitch = pow(10.0,par->pitch);

/* Quantise gain terms with uniform log quantiser   */
melp_q_gain(par->gain, par->gain_index,GN_QLO,GN_QUP,GN_QLEV);
```

```
    /* Quantise jitter and bandpass voicing */
    melp_quant_u(&par->jitter,&par->jit_index,0.0,MAX_JITTER,2);
    par->uv_flag = melp_q_bpvc(&par->bpvc[0],&par->bpvc_index,bpthresh,
                               NUM_BANDS);

    /*  Calculate Fourier coefficients of residual signal from quantised LPC */
    melp_fill(par->fs_mag,1.0,NUM_HARM);
    if (par->bpvc[0] > bpthresh) {
        melp_lpc_lsp2pred(par->lsf,lpc,LPC_ORD);
        melp_zerflt(&speech[(FRAME_END-(LPC_FRAME/2))],lpc,sigbuf,
                LPC_ORD,LPC_FRAME);
        melp_window(sigbuf,melp_win_cof,sigbuf,LPC_FRAME);
        melp_find_harm(sigbuf, par->fs_mag, par->pitch, NUM_HARM, LPC_FRAME);
    }

    /* quantize Fourier coefficients */
    /* pre-weight vector, then use Euclidean distance */
    melp_window(&par->fs_mag[0],w_fs,&par->fs_mag[0],NUM_HARM);
    melp_fsvq_enc(&par->fs_mag[0], &par->fs_mag[0], fs_vq_par);

    /* Set MELP indeces to point to same array */
    par->fsvq_index = fs_vq_par.indices;

    /* Update MSVQ information */
    par->msvq_stages = vq_par.num_stages;
    par->msvq_bits = vq_par.num_bits;

    /* Write channel bitstream */
    melp_chn_write(par);

    /* Update delay buffers for next frame */
    if (framemode == 1)          // RM, 07/20/98
        melp_v_cmult(&speech[IN_BEG+FRAME-INFRAME+FRAME],sqrttukeyend,76);
    melp_v_equ(&speech[0],&speech[FRAME],IN_BEG);
    fpitch[BEGIN] = fpitch[END];
}



/*
 * melp_enc_init: perform initialization
 */


void melp_enc_init(void)
{

    int j;

    melp_bpvc_ana_init(FRAME,PITCHMIN,PITCHMAX,NUM_BANDS,2,MINLENGTH);
    melp_pitch_ana_init(PITCHMIN,PITCHMAX,FRAME,LPF_ORD,MINLENGTH);
    melp_p_avg_init(PDECAY,DEFAULT_PITCH,3);

    melp_v_zap(speech,IN_BEG+FRAME);
    melp_v_zap(speech_lpc,IN_BEG+FRAME);
    melp_v_zap(speech_pitch,IN_BEG+FRAME);
    pitch_avg=DEFAULT_PITCH;
    melp_fill(fpitch,DEFAULT_PITCH,2);
    melp_v_zap(lpfsp_del,LPF_ORD);

    /* Initialise multi-stage vector quantisation (read codebook) */

    vq_par.num_best = MSVQ_M;
```

02/05/99
16:16:06

melp_ana.c.

6

```
  vq_par.num_stages = 4;
  vq_par.dimension = 10;

  /*
   * Allocate memory for number of levels per stage and indices
   * and for number of bits per stage
   */

  MEM_ALLOC(MALLOC,vq_par.num_levels,vq_par.num_stages,int);
  MEM_ALLOC(MALLOC,vq_par.indices,vq_par.num_stages,int);
  MEM_ALLOC(MALLOC,vq_par.num_bits,vq_par.num_stages,int);

  vq_par.num_levels[0] = 128;
  vq_par.num_levels[1] = 64;
  vq_par.num_levels[2] = 64;
  vq_par.num_levels[3] = 64;

  v~ par.num_bits[0] = 7;
     er.num_bits[1] = 6;
   _par.num_bits[2] = 6;
  vq_par.num_bits[3] = 6;

  vq_par.cb = melp_msvq_cb;

  /* Scale codebook to 0 to 1 */
  melp_v_scale(vq_par.cb,(2.0/FSAMP),3200);

  /* Initialize Fourier magnitude vector quantization (read codebook) */

  fs_vq_par.num_best = 1;
  fs_vq_par.num_stages = 1;
  fs_vq_par.dimension = NUM_HARM;

  /*
   * Allocate memory for number of levels per stage and indices
   * and for number of bits per stage
   */

  MEM_ALLOC(MALLOC,fs_vq_par.num_levels,fs_vq_par.num_stages,int);
  MEM_ALLOC(MALLOC,fs_vq_par.indices,fs_vq_par.num_stages,int);
  MEM_ALLOC(MALLOC,fs_vq_par.num_bits,fs_vq_par.num_stages,int);

  '  ~q_par.num_levels[0] = FS_LEVELS;
     q_par.num_bits[0] = FS_BITS;
  ,  q_par.cb = melp_fsvq_cb;

  /* Initialize fixed MSE weighting and inverse of weighting */

  melp_vq_fsw(w_fs, NUM_HARM, 60.0);

  /* Pre-weight codebook (assume single stage only) */

  if (melp_fsvq_weighted == 0)
    {
        melp_fsvq_weighted = 1;
        for (j = 0; j < fs_vq_par.num_levels[0]; j++)
          melp_window(&fs_vq_par.cb[j*NUM_HARM], w_fs,
               &fs_vq_par.cb[j*NUM_HARM], NUM_HARM);
```

16:16:30

mat_lib.c.

```
4 kbps MELP Federal Standard speech coder

rsion 1.2

pyright (c) 1996, Texas Instruments, Inc.

shu Viswanathan
rsonal Systems Laboratory
rporate R&D
xas Instruments
O. Box 655303, M/S 8374
llas, TX 75265

e Mixed Excitation Linear Prediction (MELP) speech coding algorithm
cluding the C source code software, the pre-existing MELP software and any
he   ments thereto, is delivered to the Government in accordance with the
     ent of Contract MDA904-94-C-6101.  It is delivered with Government
   se License Rights in the field of secure voice communications only.  No
her use is authorized or granted by Texas Instruments Incorporated. The
vernment Purpose license rights shall be effective until 30 September 2001;
ereafter, the Government purpose license rights will expire and the
vernment shall have unlimited rights in the software.  The restrictions
verning use of the software marked with this legend are set forth in the
finition of 'Government Purpose License Rights' in paragraph (a)(14) of the
ause at 252.227-7013 of the contract listed above.   This legend, together
h the indications of the portions of this software which are subject to
vernment purpose license rights shall be included on any reproduction hereof
ich includes any part of the portions subject to such limitations.



mat_lib.c: Matrix and vector manipulation library



nclude "spbstd.h"
nclude "mat.h"

 V    - vector addition */
oa    elp_v_add(Float *v1,Float *v2,int n)

  int i;

  for(i=0; i < n; i++)
      v1[i] += v2[i];
  return(v1);



 V_EQU- vector equate */
oat *melp_v_equ(Float *v1,Float *v2,int n)

  int i;

  for(i=0; i < n; i++)
      v1[i] = v2[i];
  return(v1);

 *melp_v_equ_int(int *v1,int *v2,int n)

  int i;
```

```
    for(i=0; i < n; i++)
        v1[i] = v2[i];
    return(v1);
}

/* V_INNER- inner product */
Float melp_v_inner(Float *v1,Float *v2,int n)
{
    int i;
    Float innerprod;

    for(i=0,innerprod=0.0; i < n; i++)
        innerprod += v1[i] * v2[i];
    return(innerprod);
}

/* melp_v_magsq - sum of squares */

Float melp_v_magsq(Float *v,int n)
{
    int i;
    Float magsq;

    for(i=0,magsq=0.0; i < n; i++)
        magsq += v[i] * v[i];
    return(magsq);
} /* V_MAGSQ */

/* V_SCALE- vector scale */
Float *melp_v_scale(Float *v,Float scale,int n)
{
    int i;

    for(i=0; i < n; i++)
        v[i] *= scale;
    return(v);
}

/* V_CMULT - componentwise vector multiplication, RM 07/20/98*/
Float *melp_v_cmult(Float *v1,Float *v2,int n)
{
    int i;

    for(i=0; i < n; i++)
        v1[i] *= v2[i];
    return(v1);
}

/* V_CDIV - componentwise vector division, RM 07/20/98*/
Float *melp_v_cdiv(Float *v1,Float *v2,int n)
{
    int i;

    for(i=0; i < n; i++)
        v1[i] /= v2[i];
    return(v1);
}

/* V_SUB- vector difference */
Float *melp_v_sub(Float *v1,Float *v2,int n)
{
    int i;

    for(i=0; i < n; i++)
```

02/05/99
16:16:30

mat_lib.c.

2

```
        v1[i] -= v2[i];
    return(v1);

* melp_v_zap - clear vector */

loat *melp_v_zap(float *v,int n)

    int i;

    for(i=0; i < n; i++)
        v[i] = 0.0;
    return(v);
 /* V_ZAP */

nt *melp_v_zap_int(int *v,int n)

        i;

    for(i=0; i < n; i++)
        v[i] = 0;
    return(v);
 /* V_ZAP */
```

02/05/99
16:16:50

dsp_sub.c.

```
4 kbps MELP Federal Standard speech coder

-rsion 1.2

-pyright (c) 1996, Texas Instruments, Inc.

-shu Viswanathan
-rsonal Systems Laboratory
-rporate R&D
-xas Instruments
-  Box 655303, M/S 8374
-llas, TX 75265

-is Mixed Excitation Linear Prediction (MELP) speech coding algorithm
-cluding the C source code software, the pre-existing MELP software and any
-h-   -ments thereto, is delivered to the Government in accordance with the
      -ent of Contract MDA904-94-C-6101.  It is delivered with Government
-,  -e License Rights in the field of secure voice communications only.  No
-her use is authorized or granted by Texas Instruments Incorporated. The
-vernment Purpose license rights shall be effective until 30 September 2001;
-ereafter, the Government purpose license rights will expire and the
-vernment shall have unlimited rights in the software.  The restrictions
-verning use of the software marked with this legend are set forth in the
-finition of "Government Purpose License Rights" in paragraph (a)(14) of the
-ause at 252.227-7013 of the contract listed above.   This legend, together
-th the indications of the portions of this software which are subject to
-vernment purpose license rights shall be included on any reproduction hereof
-hich includes any part of the portions subject to such limitations.



 dsp_sub.c: general subroutines.



*  compiler include files  */
-nclude         <stdio.h>
-nclude         <stdlib.h>
-nc    l        <math.h>
-nc      "dsp_sub.h"
-nclude "spbstd.h"
-nclude "mat.h"

-pedef short SPEECH;
-efine PRINT 1

*                                                          */
*      Subroutine autocorr: calculate autocorrelations     */
*                                                          */
-id melp_autocorr(float input[], float r[], int order, int npts)

   int i;

   for (i = 0; i <= order; i++ )
     r[i] = melp_v_inner(&input[0],&input[i],(npts-i));
   if (r[0] < 1.0)
    -r[0] = 1.0;


                                                           */
```

```
/*      Subroutine envelope: calculate time envelope of signal. */
/*      Note: the delay history requires one previous sample    */
/*      of the input signal and two previous output samples.    */
/*                                                              */
#define C2 (-0.9409)
#define C1 1.9266

void melp_envelope(float input[], float prev_in, float output[], int npts)

{
    int i;
    float curr_abs, prev_abs;

    prev_abs = fabs(prev_in);
    for (i = 0; i < npts; i++) {
        curr_abs = fabs(input[i]);
        output[i] = curr_abs - prev_abs + C2*output[i-2] + C1*output[i-1];
        prev_abs = curr_abs;
    }
}

/*                                                              */
/*  Subroutine fill: fill an input array with a value.          */
/*                                                              */
void melp_fill(float output[], float fillval, int npts)
{
  int i;

  for (i = 0; i < npts; i++ )
    output[i] = fillval;

}

/*                                                              */
/*      Subroutine interp_array: interpolate array              */
/*                                                              */
void melp_interp_array(float prev[], float curr[], float out[],
        float ifact, int size)

{
    int i;
    float ifact2;

    ifact2 = 1.0 - ifact;
    for (i = 0; i < size; i++)
      out[i] = ifact*curr[i] + ifact2*prev[i];

}

/*                                                              */
/*      Subroutine median: calculate median value               */
/*                                                              */
#define MAXSORT 5
float melp_median(float input[], int npts)
{
    int i,j,loc;
    float insert_val;
    float sorted[MAXSORT];

    /* sort data in temporary array */

#ifdef PRINT
    if (npts > MAXSORT) {
        printf("ERROR: median size too large.\n");
        exit(1);
  }
ndif

  melp_v_equ(sorted,input,npts);
  for (i = 1; i < npts; i++) {

      /* for each data point */
      for (j = 0; j < i; j++) {

          /* find location in current sorted list */
          if (sorted[i] < sorted[j])
            break;
      }

      /* insert new value */
      loc = j;
      insert_val = sorted[i];
      for (j = i; j > loc; j--)
        sorted[j] = sorted[j-1];
      sorted[loc] = insert_val;
  }

  return(sorted[npts/2]);


ndef MAXSORT

                                                          */
      Subroutine PACK_CODE: Pack bit code into channel.   */
                                                          */
id melp_pack_code(int code, unsigned int **p_ch_beg, int *p_ch_bit,
      int numbits, int wsize)


  int i,ch_bit;
  unsigned int *ch_word;

      ch_bit = *p_ch_bit;
      ch_word = *p_ch_beg;

      for (i = 0; i < numbits; i++) {
              /* Mask in bit from code to channel word      */
              if (ch_bit == 0)
                *ch_word = ((code & (1<<i)) >> i);
              else
                *ch_word |= (((code & (1<<i)) >> i) << ch_bit);

              /* Check for full channel word                */
              if (++ch_bit >= wsize) {
                      ch_bit = 0;
                      (*p_ch_beg)++ ;
                      ch_word++ ;
              }
      }

      /* Save updated bit counter      */
      *p_ch_bit = ch_bit;


                                                          */
      Subroutine peakiness: estimate peakiness of input   */
      signal using ratio of L2 to L1 norms.               */
                                                          */
oat melp_peakiness(Float input[], int npts)
```

```
{
    int i;
    Float sum_abs, peak_fact;

    sum_abs = 0.0;
    for (i = 0; i < npts; i++)
      sum_abs += fabs(input[i]);

    if (sum_abs > 0.01)
      peak_fact = sqrt(npts*melp_v_magsq(input,npts)) / sum_abs;
    else
      peak_fact = 0.0;

    return(peak_fact);
}

/*                                                               */
/*      Subroutine polflt: all pole (IIR) filter.                */
/*      Note: The filter coefficients represent the              */
/*      denominator only, and the leading coefficient            */
/*      is assumed to be 1.                                      */
/*      The output array can overlay the input.                  */
/*                                                               */
void melp_polflt(Float input[], Float coeff[], Float output[],
        int order,int npts)

{
    int i,j;
    Float accum;

    for (i = 0; i < npts; i++ ) {
        accum = input[i];
        for (j = 1; j <= order; j++ )
            accum -= output[i-j] * coeff[j];
        output[i] = accum;
    }
}

/*                                                               */
/*      Subroutine QUANT_U: quantize positive input value with   */
/*      symmetrical uniform quantizer over given positive        */
/*      input range.                                             */
/*                                                               */
void melp_quant_u(Float *p_data, int *p_index, Float qmin,
        Float qmax, int nlev)

{
register int    i, j;
register Float  step, qbnd, *p_in;

        p_in = p_data;

        /*  Define symmetrical quantizer stepsize       */
        step = (qmax - qmin) / (nlev - 1);

        /*  Search quantizer boundaries                 */
        qbnd = qmin + (0.5 * step);
        j = nlev - 1;
        for (i = 0; i < j; i ++ ) {
                if (*p_in < qbnd)
                        break;
                else
                        qbnd += step;
```

02/05/99
16:16:50

dsp_sub.c.

```
    /*  Quantize input to correct level              */
    *p_in = qmin + (i * step);
    *p_index = i;


                                                      */
    Subroutine QUANT_U_DEC: decode uniformly quantized */
    value.                                            */
                                                      */
id melp_quant_u_dec(int index, Float *p_data, Float qmin, Float qmax,
    int nlev)


ister Float  step;

    /*  Define symmetrical quantizer stepsize       */
    step = (qmax - qmin) / (nlev - 1);

    /*  Decode quantized level                      */
    *p_data = qmin + (index * step);



                                                      */
    Subroutine rand_num: generate random numbers to fill */
    array using system random number generator.       */
                                                      */
id    melp_rand_num(Float output[], Float amplitude, int npts)


  int i;

  for (i = 0; i < npts; i++ ) {

      /* use system random number generator from -1 to +1 */
ifdef RAND_MAX
      /* ANSI C environment */
      output[i] = (amplitude*2.0) * ((Float) rand()*(1.0/RAND_MAX) - 0.5);
lse
      /* assume Sun OS4 */
      output[i] = amplitude * (Float) (((random() >> 16)/32767. - .5)*2);
nd

  }


                                                      */
     Subroutine READBL: read block of input data       */
                                                      */
lefine MAXSIZE 1024
nt melp_readbl(Float input[], FILE *fp_in, int size)


  int i, length;
  SPEECH     int_sp[MAXSIZE]; /*  integer input array      */

ifdef PRINT
  if (size > MAXSIZE) {
  - printf("****ERROR: read block size too large **** \n");
    exit(1);
  }
ndif

  length = fread(int_sp,sizeof(SPEECH),size,fp_in);
```

```
    for (i = 0; i < length; i++ )
      input[i] = int_sp[i];
    for (i = length; i < size; i++ )
      input[i] = 0.0;

    return length;

}
#undef MAXSIZE

/*                        RM                                  */
/*     Subroutine READBL_FLOAT: read block of float input data  */
/*                                                            */
#define MAXSIZE 1024
int melp_readbl_float(Float input[], FILE *fp_in, int size)

{
    int i, length;
    float       int_sp[MAXSIZE]; /*  integer input array          */   // RM 07/
20/98

#ifdef PRINT
    if (size > MAXSIZE) {
        printf("****ERROR: read block size too large **** \n");
        exit(1);
    }
#endif

    length = fread(int_sp,sizeof(float),size,fp_in);
    for (i = 0; i < length; i++ )
      input[i] = int_sp[i];
    for (i = length; i < size; i++ )
      input[i] = 0.0;

    return length;

}
#undef MAXSIZE


/*                                                            */
/*     Subroutine UNPACK_CODE: Unpack bit code from channel.  */
/*     Return 1 if erasure, otherwise 0.                      */
/*                                                            */
int melp_unpack_code(unsigned int **p_ch_beg, int *p_ch_bit,
        int *p_code, int numbits, int wsize, unsigned int ERASE_MASK)

{

    int ret_code;
    int i,ch_bit;
    unsigned int *ch_word;

        ch_bit = *p_ch_bit;
        ch_word = *p_ch_beg;
        *p_code = 0;
        ret_code = *ch_word & ERASE_MASK;

        for (i = 0; i < numbits; i++) {
                /* Mask in bit from channel word to code       */
                *p_code |= (((*ch_word & (1<<ch_bit)) >> ch_bit) << i);

                /* Check for end of channel word               */
                if (++ch_bit >= wsize) {
                        ch_bit = 0;
```

```
                    (*p_ch_beg)++ ;
                    ch_word++ ;
            }
        }

        /*  Save updated bit counter    */
        *p_ch_bit = ch_bit;

    /* Catch erasure in new word if read */
    if (ch_bit != 0)
      ret_code |= *ch_word & ERASE_MASK;

    return(ret_code);



        Subroutine window: multiply signal by window            */
                                                                */
                                                                */
id melp_window(Float input[], Float win_cof[], Float output[], int npts)

    int i;

    for (i = 0; i < npts; i++ )
      output[i] = win_cof[i]*input[i];



        Subroutine WRITEBL: write block of output data          */
                                                                */
                                                                */
efine MAXSIZE 1024
efine SIGMAX 32767

id melp_writebl(Float output[], FILE *fp_out, int size)


    int i;
    SPEECH      int_sp[MAXSIZE]; /*  integer input array                */
    Float temp;

fde' PRINT
        size > MAXSIZE) {
        printf("*****ERROR: write block size too large **** \n");
        exit(1);
    }
ndif

    for (i = 0; i < size; i++ ) {
        temp = output[i];
        /* clamp to +- SIGMAX */
        if (temp > SIGMAX)
          temp = SIGMAX;
        if (temp < -SIGMAX)
          temp = -SIGMAX;
        int_sp[i] = temp;

    }
    fwrite(int_sp,sizeof(SPEECH),size,fp_out);


ndef MAXSIZE

                                                                */
```

```
/*      Subroutine zerflt: all zero (FIR) filter.               */
/*      Note: the output array can overlay the input.           */
/*                                                              */
void melp_zerflt(Float input[], Float coeff[], Float output[],
        int order, int npts)

{
    int i,j;
    Float accum;

    for (i = npts-1; i >= 0; i-- ) {
        accum = 0.0;
        for (j = 0; j <= order; j++ )
            accum += input[i-j] * coeff[j];
        output[i] = accum;
    }
}
```

02/05/99
16:17:09

main.c.

```
4 kbps MELP Federal Standard speech coder

rsion 1.2

pyright (c) 1996, Texas Instruments, Inc.

shu Viswanathan
rsonal Systems Laboratory
rporate R&D
xas Instruments
O. Box 655303, M/S 8374
llas, TX 75265

le    ed Excitation Linear Prediction (MELP) speech coding algorithm
cl    g the C source code software, the pre-existing MELP software and any
han..ments thereto, is delivered to the Government in accordance with the
quirement of Contract MDA904-94-C-6101.  It is delivered with Government
rpose License Rights in the field of secure voice communications only.  No
her use is authorized or granted by Texas Instruments Incorporated. The
...rnment Purpose license rights shall be effective until 30 September 2001;
  ..after, the Government purpose license rights will expire and the
vernment shall have unlimited rights in the software.  The restrictions
verning use of the software marked with this legend are set forth in the
finition of "Government Purpose License Rights" in paragraph (a)(14) of the
ause at 252.227-7013 of the contract listed above.   This legend, together
th the indications of the portions of this software which are subject to
vernment purpose license rights shall be included on any reproduction hereof
ich includes any part of the portions subject to such limitations.


*                                                                    */
*  melp.c: Mixed Excitation LPC speech coder                         */
*                                                                    */

*  compiler include files  */
include <stdio.h>
include "melp.h"
inc    e "spbstd.h"
inc    e "mat.h"
include <fstream.h>

*  compiler constants */
define ANA_SYN 0
define ANALYSIS 1
define SYNTHESIS 2


* note: CHSIZE is shortest integer number of words in channel packet */
define CHSIZE 9
define NUM_CH_BITS 54

*  external memory */
nt melpmode = ANA_SYN;
nt framemode = 0;          // RM 07/20/98
nt autocorrmode = 0;       // RM 08/04/98
nt floatmode = 0;          // RM 08/04/98
nt writemode = 0;
nt readmode = 0;
nt filtermode = 0;
nt lpcmode = 0;
nt pitchmode = 0;
```

```
Float r1[LPC_ORD+1];


char in_name[80], in_name_lpc[80], in_name_pitch[80], out_name[80], out_params_name[80
], in_params_name[80], autocorr_name[80];

void main(int argc, char **argv)

{
        void parse(int argc, char **argv);

        int inFrame;
        Float *speech_in, *speech_in_lpc, *speech_in_pitch;

        int length, frame, eof_reached;
        int num_frames = 0;

        // Float speech_in[FRAME];                  // RM, 07/20/98
        Float speech_out[FRAME];
        static struct melp_param melp_par;      /* melp parameters */

        static struct melp_param new_par;      /* new melp parameters */

        unsigned int chan_bit[CHSIZE];
        FILE *fp_in, *fp_out, *fp_in_lpc, *fp_in_pitch;

        FILE  *fp_autocorrin;

        /* Print user message */
        printf("\n2.4 kb/s Federal Standard MELP speech coder\n");
        printf("  C simulation, version RM \n\n");

//**************** Get input parameters from command line ***********************
        parse(argc, argv);

        ofstream track1(out_params_name);
        if (!track1 && writemode == 1) { cerr << "Cannot open " << out_params_name <<
" output file!" << endl;
              exit(1);
              }

        ifstream inparams(in_params_name);
           if (!inparams && readmode == 1) {cerr << "Cannot open " << in_params_name <
< " input file!" << endl;
           exit(1);
           };


        if (autocorrmode==1) {
           if (( fp_autocorrin = fopen(autocorr_name,"rb")) == NULL ) {
                printf("  ERROR: cannot read file %s.\n",autocorr_name);
                exit(1);
           }
        };

        if (framemode==1)
          {
           inFrame = INFRAME;
           MEM_ALLOC(MALLOC,speech_in,INFRAME,Float); // RM, 07/20/98
           MEM_ALLOC(MALLOC,speech_in_lpc,INFRAME,Float);
           MEM_ALLOC(MALLOC,speech_in_pitch,INFRAME,Float);
          }
        else
          {
           inFrame = FRAME;
```

```
	MEM_ALLOC(MALLOC,speech_in,FRAME,Float);		// RM, 07/20/98
	MEM_ALLOC(MALLOC,speech_in_lpc,FRAME,Float);
	MEM_ALLOC(MALLOC,speech_in_pitch,FRAME,Float);
      };

    /* Open input, output, and parameter files */
    if (( fp_in = fopen(in_name,"rb")) == NULL ) {
	printf("  ERROR: cannot read file %s.\n",in_name);
	exit(1);
    }

    if (lpcmode ==1) {
      if (( fp_in_lpc = fopen(in_name_lpc,"rb")) == NULL ) {
	printf("  ERROR: cannot read lpc file %s.\n",in_name_lpc);
	exit(1);
      }
    };

    if (pitchmode == 1) {
      if (( fp_in_pitch = fopen(in_name_pitch,"rb")) == NULL ) {
	printf("  ERROR: cannot read pitch file %s.\n",in_name_pitch);
	exit(1);
      }
    };

    if (( fp_out = fopen(out_name,"wb")) == NULL ) {
	printf("  ERROR: cannot write file %s.\n",out_name);
	exit(1);
    }

    /* Check length of channel input if needed */
    if (melpmode == SYNTHESIS) {
	fseek(fp_in,0L,2);
	length = ftell(fp_in);
	rewind(fp_in);

	num_frames = 0.5 + length * (8.0 / NUM_CH_BITS) * (6.0/32);

    }

    int inputf_num, length_lpc, length_pitch;

    /* Check length of input file if needed, RM, 07/20/98*/
    if (melpmode != SYNTHESIS && (framemode==1)) {
	fseek(fp_in,0L,2);
	length = ftell(fp_in);
	rewind(fp_in);

	if (floatmode == 1) {
	   if ((length/sizeof(float)) % inFrame != 0) {
	      printf("  ERROR: file %s must contain a multiple of 256 samples.
",in_name);
	      exit(1);
	   };

	   inputf_num = length/sizeof(float) / inFrame;

	}
	else {
	   if ((length/sizeof(short)) % inFrame != 0) {
	      printf("  ERROR: file %s must contain a multiple of 256 samples.
",in_name);
	      exit(1);
	   };
	   inputf_num = length/sizeof(short) / inFrame;
	   };

    }
    else if (melpmode != SYNTHESIS && (framemode==0)) {
	fseek(fp_in,0L,2);
	length = ftell(fp_in);
	rewind(fp_in);

	if (floatmode == 0)
	   inputf_num = length/sizeof(short) / inFrame;
	else
	   inputf_num = length/sizeof(float) / inFrame;

    };

    if (melpmode != SYNTHESIS && (lpcmode==1)) {
	fseek(fp_in_lpc,0L,2);
	length_lpc = ftell(fp_in_lpc);
	rewind(fp_in_lpc);
	if ((length_lpc != length)) {
	   printf("  ERROR: file %s must contain the same number of samples as
file %s.\n",
					in_name_lpc,in_name);
	   exit(1);
	};
    };

    if (melpmode != SYNTHESIS && (pitchmode==1)) {
	fseek(fp_in_pitch,0L,2);
	length_pitch = ftell(fp_in_pitch);
	rewind(fp_in_pitch);
	if ((length_pitch != length)) {
	   printf("  ERROR: file %s must contain the same number of samples as
file %s.\n",
					in_name_pitch,in_name);
	   exit(1);
	};
    };

    /* Check length of autocorr input file if needed, RM, 08/04/98*/
    if (melpmode != SYNTHESIS && (autocorrmode==1)) {

	fseek(fp_autocorrin,0L,2);
	length = ftell(fp_autocorrin);
	rewind(fp_autocorrin);

	if ((length/sizeof(float)) % (LPC_ORD+1) != 0)
	  {
	  printf("  ERROR: file autocorr.8k must contain a multiple of (LPC_OR
D + 1) samples.\n");
	  exit(1);
	}

	if (inputf_num+1 != length/sizeof(float) / (LPC_ORD+1))
	{
	   printf("  ERROR: file autocorr.8k must contain one input frame mor
e than %s.\n",in_name);
	   exit(1);
	}
    }

    /* Initialize MELP analysis and synthesis */
    if (melpmode != SYNTHESIS)
	melp_enc_init();
```

02/05/99
16:17:09

main.c.

3

```
      if (melpmode != ANALYSIS)
              melp_dec_init();

      /* Run MELP coder on input signal */
      frame = 0;
      eof_reached = 0;
      while (eof_reached == 0) {

              /* Perform MELP analysis */
              if (melpmode != SYNTHESIS) {

                      /* read input speech */
                if ((floatmode == 0) {
                        length = melp_readbl(speech_in,fp_in,inFrame); //RM,07/20/

                        if (lpcmode == 1)
                           melp_readbl(speech_in_lpc,fp_in_lpc,inFrame);
                        if (pitchmode == 1)
                           melp_readbl(speech_in_pitch,fp_in_pitch,inFrame);
                }
                else {
                        length = melp_readbl_float(speech_in,fp_in,inFrame); //RM,
1/06/98
                        if (lpcmode == 1)
                           melp_readbl_float(speech_in_lpc,fp_in_lpc,inFrame);
                        if (pitchmode == 1)
                           melp_readbl_float(speech_in_pitch,fp_in_pitch,inFrame);
                };

                if (autocorrmode == 1)
                          melp_readbl_float(r1,fp_autocorrin,LPC_ORD+1);

                if (length < inFrame) {
                    melp_v_zap(&speech_in[length],inFrame-length);
                    if (lpcmode == 1)
                          melp_v_zap(&speech_in_lpc[length],inFrame-length);
                    if (pitchmode == 1)
                          melp_v_zap(&speech_in_pitch[length],inFrame-length);
                    eof_reached = 1;
                    }

                  if (writemode == 1) {
                  if ((lpcmode == 0) && (pitchmode == 0))
                      melp_enc(speech_in, speech_in, speech_in, chan_bit, &melp
ar);

                  if ((lpcmode == 1) && (pitchmode == 0))
                      melp_enc(speech_in, speech_in_lpc, speech_in, chan_bit, &
lp_par);

                  if ((lpcmode == 0) && (pitchmode == 1))
                      melp_enc(speech_in, speech_in, speech_in_pitch, chan_bit,    ar, &new_par);
melp_par);

                  if ((lpcmode == 1) && (pitchmode == 1))
                      melp_enc(speech_in, speech_in_lpc, speech_in_pitch, chan_
t, &melp_par);                                                                   _par);

                    track1 << double (melp_par.pitch) << " "
                       << melp_par.pitch_index    << " "                        melp_par);
                       << double (melp_par.bpvc[0]) << " "
                       << double (melp_par.bpvc[1]) << " "
                       << double (melp_par.bpvc[2]) << " "
                       << double (melp_par.bpvc[3]) << " "
```

```
                       << double (melp_par.bpvc[4]) << " "
                       << melp_par.uv_flag        << " "
                       << double (melp_par.gain[0]) << " "
                       << double (melp_par.gain[1]) << " "
                       << double (melp_par.jitter)  << " "
                       << double (melp_par.lsf[1])  << " "
                       << double (melp_par.lsf[2])  << " "
                       << double (melp_par.lsf[3])  << " "
                       << double (melp_par.lsf[4])  << " "
                       << double (melp_par.lsf[5])  << " "
                       << double (melp_par.lsf[6])  << " "
                       << double (melp_par.lsf[7])  << " "
                       << double (melp_par.lsf[8])  << " "
                       << double (melp_par.lsf[9])  << " "
                       << double (melp_par.lsf[10]) << " "
                       << melp_par.msvq_index[0]  << " "
                       << melp_par.msvq_index[1]  << " "
                       << melp_par.msvq_index[2]  << " "
                       << melp_par.msvq_index[3]  << endl;
                  }
                  else if (readmode == 1 && writemode == 0) {
                    double ddummy;
                    int idummy;

                    inparams >> ddummy; new_par.pitch = Float (ddummy);
                    inparams >> idummy; new_par.pitch_index = idummy;
                    inparams >> ddummy; new_par.bpvc[0] = Float (ddummy);
                    inparams >> ddummy; new_par.bpvc[1] = Float (ddummy);
                    inparams >> ddummy; new_par.bpvc[2] = Float (ddummy);
                    inparams >> ddummy; new_par.bpvc[3] = Float (ddummy);
                    inparams >> ddummy; new_par.bpvc[4] = Float (ddummy);
                    inparams >> idummy; new_par.uv_flag = idummy;
                    inparams >> ddummy; new_par.gain[0] = Float (ddummy);
                    inparams >> ddummy; new_par.gain[1] = Float (ddummy);
                    inparams >> ddummy; new_par.jitter = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[1] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[2] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[3] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[4] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[5] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[6] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[7] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[8] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[9] = Float (ddummy);
                    inparams >> ddummy; new_par.lsf[10] = Float (ddummy);

                    inparams >> idummy; // new_par.msvq_index[0] = idummy;
                    inparams >> idummy; // new_par.msvq_index[1] = idummy;
                    inparams >> idummy; // new_par.msvq_index[2] = idummy;
                    inparams >> idummy; // new_par.msvq_index[3] = idummy;

                    melp_enc(speech_in, speech_in, speech_in, chan_bit, &melp_p

                  }
                  else if (writemode == 0)  {
                    if ((lpcmode == 0) && (pitchmode == 0))
                      melp_enc(speech_in, speech_in, speech_in, chan_bit, &melp

                    if ((lpcmode == 1) && (pitchmode == 0))
                      melp_enc(speech_in, speech_in_lpc, speech_in, chan_bit, &

                    if ((lpcmode == 0) && (pitchmode == 1))
                      melp_enc(speech_in, speech_in, speech_in_pitch, chan_bit,
```

main.c.

```
.melp_par);

                if ((lpcmode == 1) && (pitchmode == 1))
                  melp_enc(speech_in, speech_in_lpc, speech_in_pitch, chan_
·t, &melp_par);

              }
              else  {      //  invalid mode
                 printf("  ERROR: invalid read/write mode! /n");
                 exit(1);
              };

              /* if ANALYSIS mode only */
              if (melpmode ==ANALYSIS && melp_par.chbit == 0) {
                     /* Write channel output to file */
                     fwrite ((void *) chan_bit, sizeof (int),
                            CHSIZE, fp_out);

              }
          }

          /* Perform MELP synthesis (skip first frame) */
          if (melpmode != ANALYSIS) {

              /* if SYNTHESIS mode only */
              if (melpmode == SYNTHESIS && melp_par.chbit == 0) {
                     /* Read channel input from file */
                     fread((void*) chan_bit, sizeof (int), CHSIZE,
                            fp_in);
              }

              melp_dec(chan_bit, speech_out, &melp_par);


              if (frame > 0)
                     melp_writebl(speech_out,fp_out,FRAME);
          }

          frame++;
          if (melpmode == SYNTHESIS) {
                 if (frame >= num_frames)
                        eof_reached = 1;
          }
      }


      fclose(fp_in);
      fclose(fp_out);


      if (trackl.bad() & writemode == 1)
         cerr << "File " << out_params_name << " had some problems." << endl;


id parse(int argc,char **argv){

      int error_flag;

      error_flag = 0;

      if (argc < 2)
            error_flag = 1;
```

```
        melpmode = ANA_SYN;

        while ((--argc>0) && ((*++argv)[0] == '-')){

                switch ((*argv)[1]){
                case 'a':
                        melpmode=ANALYSIS;
                        break;

                case 's':
                        melpmode=SYNTHESIS;
                        break;

                case 'i':
                        sscanf(*++argv,"%s",in_name);
                        --argc;
                        break;

                case 'o':
                        sscanf(*++argv,"%s",out_name);
                        --argc;
                        break;
                case 'e':                       // RM, 07/20/98  read input signal
 in frames of 256
                        framemode = 1;
                        break;
                case 'r':                       // RM, 08/04/98  read additional f
ile with autocorrelation values
                        autocorrmode = 1;
                        sscanf(*++argv,"%s",autocorr_name);
                        --argc;
                        break;
                case 'f':                       // RM, 08/06/98  read input file i
n float format
                        floatmode = 1;
                        break;
                case 'w':                       // RM, 08/06/98  write parameters
to file
                        writemode = 1;
                        sscanf(*++argv,"%s",out_params_name);
                        --argc;
                        break;
                case 'c':                       // RM, 08/06/98  write parameters
to file
                        readmode = 1;
                        sscanf(*++argv,"%s",in_params_name);
                        --argc;
                        break;
                case 'l':                       // RM, 08/24/98  read separate fil
e for lpc
                        lpcmode = 1;
                        sscanf(*++argv,"%s",in_name_lpc);
                        --argc;
                        break;
                case 'p':                       // RM, 08/24/98  read separate fil
e for pitch
                        pitchmode = 1;
                        sscanf(*++argv,"%s",in_name_pitch);
                        --argc;
                        break;
                case 'h':                       // RM, 08/06/98  apply input filte

                        filtermode = 1;
                        break;
                default:
```

02/05/99
16:17:09

main.c.

5

```
                        error_flag = 1;
                        break;
                }
        }

        if (error_flag == 1) {
            fprintf(stderr,"Usage:\n\n");
            fprintf(stderr,"Analysis/synthesis: melp -i infile -o outfile\n");
            fprintf(stderr,"Analysis only:  melp -a -i infile  -o bitfile\n");
            fprintf(stderr,"Synthesis only: melp -s -i bitfile -o outfile\n");
            fprintf(stderr,"\n");
            fprintf(stderr,"-f : speech input file is in float format.\n");
            fprintf(stderr,"-e : speech input file contains frames of 256 samples each
n");
            fprintf(stderr,"-r <filename> : read additional autocorrelation values in
nat format from file <filename>.\n");
            fprintf(stderr,"-w <filename> : Write MELP parameters to file <filename>.\
);
            fprintf(stderr,"-c <filename> : Read MELP parameters from file <filename>.
oes not work with -w option!\n");
            fprintf(stderr,"-l <filename> : Read lpc info from file <filename>.\n");
            fprintf(stderr,"-p <filename> : Read pitch info from file <filename>.\n");
            fprintf(stderr,"-h : Apply input highpass filter to input files.\n");
            exit(1);
        }

        if (melpmode == ANA_SYN)
                printf(" MELP analysis and synthesis \n");
        else
                if (melpmode == ANALYSIS)
                        printf(" MELP analysis \n");
                else
                        if (melpmode == SYNTHESIS)
                                printf(" MELP synthesis \n");
        if (framemode==1)
                printf(" --> Read enhanced frames of size INFRAME. \n");

        printf("   input from %s\n   output to %s.\n",in_name, out_name);
```

```
4 kbps MELP Federal Standard speech coder

ersion 1.2

:pyright (c) 1996, Texas Instruments, Inc.

ishu Viswanathan
rsonal Systems Laboratory
rporate R&D
xas Instruments
). Box 655303, M/S 8374
allas, TX 75265

his Mixed Excitation Linear Prediction (MELP) speech coding algorithm
ir   'ng the C source code software, the pre-existing MELP software and any
ih    ments thereto, is delivered to the Government in accordance with the
equirement of Contract MDA904-94-C-6101.  It is delivered with Government
urpose License Rights in the field of secure voice communications only.  No
ther use is authorized or granted by Texas Instruments Incorporated. The
overnment Purpose license rights shall be effective until 30 September 2001;
hereafter, the Government purpose license rights will expire and the
overnment shall have unlimited rights in the software.  The restrictions
overning use of the software marked with this legend are set forth in the
efinition of 'Government Purpose License Rights' in paragraph (a)(14) of the
lause at 252.227-7013 of the contract listed above.   This legend, together
ith the indications of the portions of this software which are subject to
overnment purpose license rights shall be included on any reproduction hereof
hich includes any part of the portions subject to such limitations.

/

 dsp_sub.h: include file

/

ifndef __FLOAT__
ypedef double Float;
h      __FLOAT__
if

* External function definitions */
oid melp_autocorr(Float input[], Float r[], int order, int npts);

oid melp_envelope(Float input[], Float prev_in, Float output[], int npts);

oid melp_fill(Float output[], Float fillval, int npts);

oid melp_interp_array(Float prev[],Float curr[],Float out[],Float ifact,int size);

loat melp_median(Float input[], int npts);

oid  melp_pack_code(int code,unsigned int **p_ch_beg,int *p_ch_bit,int numbits,
      int size);

loat melp_peakiness(Float input[], int npts);

id melp_polflt(Float input[], Float coeff[], Float output[], int order,int npts);

id melp_quant_u(Float *p_data, int *p_index, Float qmin, Float qmax, int nlev);

id melp_quant_u_dec(int index, Float *p_data,Float qmin, Float qmax, int nlev);
```

```
void melp_rand_num(Float output[],Float amplitude, int npts);

int melp_unpack_code(unsigned int **p_ch_beg, int *p_ch_bit, int *p_code,
        int numbits, int wsize, unsigned int erase_mask);

void melp_window(Float input[], Float win_cof[], Float output[], int npts);

void melp_zerflt(Float input[], Float coeff[], Float output[], int order,int npts);

int melp_readbl(Float input[], FILE *fp_in, int size);
int melp_readbl_float(Float input[], FILE *fp_in, int size);    /* RM */

void melp_writebl(Float output[], FILE *fp_out, int size);
```

16:18:05

mat.h.

[illegible]

```
( kbps MELP Federal Standard speech coder

raion 1.2

pyright (c) 1996, Texas Instruments, Inc.

shu Viswanathan
rsonal Systems Laboratory
rporate R&D
xas Instruments
O. Box 655303, M/S 8374
llas, TX 75265

is Mixed Excitation Linear Prediction (MELP) speech coding algorithm
cluding the C source code software, the pre-existing MELP software and any
har  ments thereto, is delivered to the Government in accordance with the
m     ent of Contract MDA904-94-C-6101.  It is delivered with Government
rp    License Rights in the field of secure voice communications only.  No
her use is authorized or granted by Texas Instruments Incorporated. The
vernment Purpose license rights shall be effective until 30 September 2001;
ereafter, the Government purpose license rights will expire and the
vernment shall have unlimited rights in the software.  The restrictions
verning use of the software marked with this legend are set forth in the
finition of 'Government Purpose License Rights' in paragraph (a)(14) of the
ause at 252.227-7013 of the contract listed above.  This legend, together
th the indications of the portions of this software which are subject to
vernment purpose license rights shall be included on any reproduction hereof
ich includes any part of the portions subject to such limitations.


 mat.h     Matrix include file.
           (Low level matrix and vector functions.)

 Copyright (c) 1995 by Texas Instruments, Inc.  All rights reserved.


fndef __FLOAT__
pedef double Float;
ef      __FLOAT__
nd

oat melp_v_inner(Float *v1,Float *v2,int n);
oat melp_v_magsq(Float *v,int n);
oat *melp_v_zap(Float *v,int n);
oat *melp_v_equ(Float *v1,Float *v2,int n);
oat *melp_v_sub(Float *v1,Float *v2,int n);
oat *melp_v_add(Float *v1,Float *v2,int n);
oat *melp_v_scale(Float *v,Float scale,int n);
t  *melp_v_zap_int(int *v,int n);
t  *melp_v_equ_int(int *v1,int *v2,int n);

oat *melp_v_cdiv(Float *v1,Float *v2,int n);
oat *melp_v_cmult(Float *v1,Float *v2,int n);
```

02/05/99
15:58:36

enhance.c.

```
/*---------------------------------------------------------------
 * enhance.c - Main program for MMSE-LSA speech enhancement with
 *             minimum statistics noise estimation
 *
 * Author: Rainer Martin, AT&T Labs-Research
 *
 * Last Update: $Id:$
 *
 *---------------------------------------------------------------
 */

include "enhance.h"
include "vect_fun.h"
include "enh_fun.h"

static char in_name[80], out_name[80], version_name[40];

...    in(int argc,  char *argv[])

 int i, lread;

 Enhance_Params P;
 Enhance_Data D;

 FILE *fp_in, *fp_out;
 float *speech_in_float, *speech_out_float, *speech_overlap_float, *new_speech;
 short *speech_short, *noises;

 /* Get input parameters from command line and overwrite default parameters if necess
ary*/
 parse_command_line(argc,argv);
 init_params(&P,version_name);   /* initialize parameters with default values*/

 speech_short = CALLOC_SHORT(P.win_shift);          /* buffer for new input samples */
 speech_in_float = CALLOC_FLOAT(P.win_len);         /* buffer of input samples of one
frame */
 speech_out_float = CALLOC_FLOAT(P.win_len);        /* buffer of input samples of one
frame */
 speech_overlap_float = CALLOC_FLOAT(P.win_len);    /* buffer of input samples of one
frame */
 new_speech = &speech_in_float[P.overlap_len];      /* pointer on new data */

 n    s = CALLOC_SHORT(P.noise_frames*P.win_shift+P.win_shift);  /* noise buffer */

 /* Print user message */
 printf("\nMMSE-LSA speech enhancement with Minimum Statistics noise estimation.\n");
 printf("  C simulation, version 1.0\n\n");
 printf("\tinput from:\t %s\n\toutput to:\t %s\n\tversion:\t %s\n",
                      in_name, out_name , version_name);

 /* Open input, output, and parameter files */
 if (( fp_in = fopen(in_name,"rb")) == NULL ) {
   printf("  ERROR: cannot read file %s. (enhance)\n",in_name);
   exit(1);
 }

 if (( fp_out = fopen(out_name,"wb")) == NULL ) {
   printf("  ERROR: cannot write file %s. (enhance)\n",out_name);
   exit(1);
 }

 /* peek at first frames for noise estimate */
 D.initial_noise = CALLOC_FLOAT(P.noise_frames*P.win_shift+P.win_shift);
 fread( (void *) noises, sizeof(short), P.noise_frames*P.win_shift+P.win_shift, fp_in
 );
 for ( i = 0; i < P.noise_frames*P.win_shift+P.win_shift; i++ ) {
   D.initial_noise[i] = (Float) noises[i];
 }

 fseek( fp_in, 0, SEEK_SET );

 enh_init(&D,&P);   /* initialize enhancement routine */

 /* Read in overlap_len of speech data */
 fread((void *) speech_short, sizeof(short), P.overlap_len, fp_in);
 for (i = 0; i <  P.overlap_len; i++)
   speech_in_float[i] = (Float) speech_short[i];

 /* main processing loop */
 while((lread = fread((void *)speech_short, sizeof(short), P.win_shift, fp_in)) > 0)
   {
    for (i = lread; i < P.win_shift; i++) /* add zeros at end of file  */
      speech_short[i] = 0;

    /* read in new speech samples and cast to Float */
    for (i = 0; i < P.win_shift; i++)
      new_speech[i] = (Float)speech_short[i];

    /* enhance one frame of (noisy) speech */
    process_frame(speech_in_float, speech_out_float, &D, &P);

    /* overlap add output buffer: move half-cooked samples to the beginning of the buf
fer,
       add overlapping buffer section, and write remaining section in output buffer */
    vec_copy(speech_overlap_float,speech_overlap_float+P.win_shift,P.overlap_len);
    vec_accu(speech_overlap_float,speech_out_float,P.overlap_len);
    vec_copy(speech_overlap_float+P.overlap_len,speech_out_float+P.overlap_len,P.win_s
hift);

    /*    for ( i = 0; i < P.win_len; i++ )
             fprintf(stdout,"%d \t %16.16f \t %16.16f \n", i+1, speech_in_float[i], speec
h_overlap_float[i]); */

    /* shift input buffer for next frame */
    vec_copy(speech_in_float,speech_in_float+P.win_shift,P.overlap_len);

#ifdef WRITEFLOAT
    /* write to file */
    fwrite( (void *) speech_overlap_float, sizeof(Float), P.win_shift, fp_out );
#else
    /* conversion to short with arithmetik rounding (instead of truncation) */
    float_to_short(speech_overlap_float,speech_short,P.win_shift);

    /* write to file */
    fwrite( (void *) speech_short, sizeof(short), P.win_shift, fp_out );
#endif
    }

   /* free memory */
   enh_terminate(&D,&P);

   free(speech_in_float);
   free(speech_out_float);
   free(speech_overlap_float);
   free(speech_short);
   free(noises);
   free(D.initial_noise);
}
```

```
id parse_command_line(int argc,char **argv)

int error_flag;

rror_flag = 0;

f (argc < 7)
  error_flag = 1;

while ((--argc>0) && ((*++argv)[0] == '-') && (error_flag == 0)){

  switch ((*argv)[1]){

  case 'i':
    sscanf(*++argv,"%s",in_name);
    --argc;
      sak;

  case 'o':
    sscanf(*++argv,"%s",out_name);
    --argc;
    break;

  case 'v':
    sscanf(*++argv,"%s",version_name);
    --argc;
    break;

  default:
    error_flag = 1;
    break;
  }
}

if (error_flag == 1) {
  fprintf(stderr,"Speech Enhancement Preprocessor Usage:\n\n");
  fprintf(stderr,"enhance -i <input_file> -o <output_file> -v version\nversion is on
of {nsa6|nsa7|nsa8} \n\n");
  exit(1);
}
```

02/05/99
15:59:08

enhance.h

```
indef __enhance__
-fine __enhance__
   ........ ... .. .  .. ..    . ............. . .... ................ ....
enhance.h - Data Structures For All Enhancement Routines

Author: Rainer Martin, AT&T Labs-Research

Last Update: $Id:$

  ..................... .. ........ ............. .........................
/

nclude "globals.h"
nclude "fftreal.h"

******************************* PARAMETERS ********************************/
p    struct noise_params_Malah {

int wtr_front_len;

float hear_thr_rms;
float env_rate;
float alpha_env;
float beta_env;
float reen_thr;
float GM_MIN;

float f_n_min;           /* Min. Noise-update factor - Noise only */
float f_n_max;           /* Max. Noise-update factor - Noise only */
float f_s_min;           /* Min. Noise-update factor - Signal present */
float f_s_max;           /* Max. Noise-update factor - Signal present */

float nsmth_bias;        /* bias factor for initial noise estimate */
float noise_b_by_nsmth_b;

float GAMAV_TH;         /* Gamma_av upper threshold */

float gammax_thr;      /* Gamma_max threshold */
float gammav_thr;        /* Gamma_av threshold */
float gammaxs_thr;       /* Gamma threshold - signal present */

No.. _Params_Malah;


pedef struct noise_params_minstat {

int wtr_front_len;
float reen_thr;
float hear_thr_rms;

float GM_MIN;

float nsmth_bias;        /* bias factor for initial noise estimate */
float noise_b_by_nsmth_b;

float GAMAV_TH;         /* Gamma_av upper threshold */

float gammax_thr;      /* Gamma_max threshold */
float gammav_thr;        /* Gamma_av threshold */

int len_minwin;         /* length of subwindow */
int num_minwin;         /* number of subwindows */
int Dmin;               /* total length of window for minimum search */
```

```
 float alpha_N_max;     /* maximum of time varying smoothing parameter */
 float Pdecay_num;

 float minv;
 float minv_sub;
 float Pvar;
 float Pvar_sub;

} Noise_Params_Minstat;


typedef struct enhance_params {

 int ENVLP_FLG;         /* Set to 0 for no lower_envelope tracking
                           Set to 1 for tracking with Spectral Sampling
                           Set to 2 for tracking with Mean Matching only */

 int NS_FLG;            /* Set to 1 to update noise_spec - signal present */

 int fs;
 int win_len;
 float win_len_inv;
 int win_shift;
 int overlap_len;
 float win_ratio;
 float win_shift_ratio;
 int vec_lenf;
 float rate;
 float rate_factor;

 int noise_frames;
 float noise_bias;
 float gN;

 float alpha_LT;
 float beta_LT;

 float qk_max;           /* Max value for prob. of signal absence */
 float qk_min;           /* Min value for prob. of signal absence */

 float alphak;           /* decision directed ksi weight */
 float betak;

 float alphay;           /* YY recursive (inter_frame) smth factor */
 float betay;

 float ksi_minq;
 float vb;               /*  'false alarm' prob. for setting thld to find qk's  */
 float gammaq_thr;

 float alphaq;           /* Smoothing factor of hard-decision qk-value  */
 float betaq;

 int software_ver;
 float* analysis_window;

#ifdef MALAH
 Noise_Params_Malah *NP;
#else
 Noise_Params_Minstat *NP;
#endif

} Enhance_Params;
```

02/05/99
15:59:08

enhance.h.

```
 Enhancement state information */
def struct enhance_data {
nt I;
loat *initial_noise;
loat *qk;
loat *qla;
loat *lambdaD;
loat *YY0;
loat *Agal;
loat *tempV1;
loat *tempV2;
loat *tempV3;
loat *analy;
loat *Y;
loat *YY;
loat *Ymau;
l     *gamaK;
l.     kal;
loat *Gain;
loat *GM;
loat *GainD;
loat *vk;
loat *ygal;

loat N_pwr;
loat GM_min;
loat Kal_min;
loat Kal_min_var;

loat YY_LT;
loat SN_LT;
loat SN_LT0;

ltr fcache;

loat n_pwr;
loat ndiff;

def MALAH
/* only for Malah's noise estimation */
loat *YY_smth;
n     c10;
n     ;01;
loat *EY;
loat YY0_av;
loat EY_av;
loat N_pwr0;

loat envlp;
nt env_flg;
nt env_drop_flg;
nt updn_flg;
nt YY_flg;
se

/* only for minimum statistics */

loat *smoothedspect;
loat *biased_smoothedspect;
loat *biased_smoothedspect_sub;
loat *circb_min;
loat *act_min;
loat *act_min_sub;
loat *noisespect;
loat* alpha_var;
```

```
  Float **circb;                    /* ring buffer */
  int circb_indx;                   /* ring buffer pointer */

  short *localflag;
  int minspec_counter;
  Float alphacorr;

  Float *var_sp_av;
  Float *var_sp_2;
  Float *var_rel;
  Float var_rel_av;
#endif

} Enhance_Data;

void parse_command_line(int argc, char **argv);


#endif
```

02/05/99
15:59:28

enh_fun.c

```
..lude "globals.h"
..lude "enh_fun.h"
.lude "enhance.h"
..lude "vect_fun.h"
..lude "windows.h"
..lude <string.h>
..lude <float.h>



   ..  . ..... . ...       .           . ...... ..  .    .  .   .......
*     un.c - Speech Enhancement Functions

Author: Rainer Martin, AT&T Labs Research

Last Update: $Id:$

    ................... .. .... .     . .    .... ... ... .............


.................................................................../
      Subroutine CALLOC_FLOAT: memory allocation for Float vectors        */
..................................................................../
it* CALLOC_FLOAT(int num_samples)

loat* tmp;
mp = calloc(num_samples,sizeof(Float));
.f (tmp == NULL)
  {
   printf("\nERROR: CALLOC_FLOAT request cannot be satisfied! \n\n");
   terminate(1);
  };

.eturn tmp;


..     ............................................................../
      Subroutine CALLOC_FLOATP: memory allocation for pointers to         */
                                 Float vectors                            */
..................................................................../
..t** CALLOC_FLOATP(int num_samples)

float** tmp;
tmp = calloc(num_samples,sizeof(Float*));
.f (tmp == NULL)
  {
   printf("\nERROR: CALLOC_FLOATP request cannot be satisfied! \n\n");
   terminate(1);
  };

.eturn tmp;


..................................................................../
      Subroutine CALLOC_SHORT: memory allocation for short vectors        */
..................................................................../
.rt* CALLOC_SHORT(int num_samples)
```

```
{
  short* tmp;
  tmp = calloc(num_samples,sizeof(short));
  if (tmp == NULL)
    {
     printf("\nERROR: CALLOC_SHORT request cannot be satisfied! \n\n");
     terminate(1);
    };

  return tmp;

}

/**************************************************************************/
/*      Subroutine terminate: terminate enhancement program              */
/**************************************************************************/

void terminate(int error_no)
{
  printf("Program exit with error code: %d\n\n",error_no);
  exit(1);
}

#ifdef MALAH
/**************************************************************************/
/*      Subroutine init_noise_params_malah: initialize parameters of noise  */
/*                              estimation procedure                      */
/**************************************************************************/
void init_noise_params_malah(Enhance_Params* p)
{
  p->NP->str_front_len = 32;

  p->NP->hear_thr_rms = 6.0;                 /* hearing thld RMS */
  p->NP->env_rate = 1. + 0.02 * p->win_ratio; /* lower envelope rise in dB */
  p->NP->alpha_env = 1. - 1e-4 * p->win_ratio * p->win_shift_ratio; /* lower envelope
parameter */
  p->NP->beta_env = 1. - p->NP->alpha_env;
#ifdef USEDOUBLES
  p->NP->resn_thr = 20 * log10 (p->NP->hear_thr_rms); /* desired residual abs noise le
vel */
#else
  p->NP->resn_thr = 20 * log10f (p->NP->hear_thr_rms); /* desired residual abs noise l
evel */
#endif

  p->NP->GM_MIN= 0.12;   /* max value for Min. Gain Mutlf  Factor - GM_min */

  p->NP->f_n_min = 0.01 * p->win_ratio * p->win_shift_ratio; /* Min. Noise-update fact
or - Noise only */
  p->NP->f_n_max = 0.1 * p->win_ratio * p->win_shift_ratio; /* Max. Noise-update facto
r - Noise only */
  p->NP->f_s_min = 0.02 * p->win_ratio * p->win_shift_ratio; /* Min. Noise-update fact
or - Signal present*/
  p->NP->f_s_max = 0.10 * p->win_ratio * p->win_shift_ratio; /* Max. Noise-update fact
or - Signal present */

  p->NP->nsmth_bias = 1 + ( 1 - p->win_ratio ) / 3.0;   /* bias factor for initial noi
se estimate */
  p->NP->noise_b_by_nsmth_b = p->noise_bias * p->NP->nsmth_bias;

#ifdef USEDOUBLES
  p->NP->GAMAV_TH = 2. * log ( 2. );            /* Upper thld on gamma_av for noise on
ly cond. */
#else
  p->NP->GAMAV_TH = 2. * logf ( 2. );
```

```
ndif

p->NP->gammav_thr = 2*p->gN;
p->NP->gammax_thr = 50.0*p->gN;  /* 10 for stationary noise, 18 for babble */
p->NP->gammaxs_thr = 3.6 - ( 2 - p->win_ratio ) * 0.6;

ndif
fdef MINSTAT
*****************************************************************************/
      Subroutine minscaling: compute scaling factor for the approximation of */
                  the bias of the minimum power estimate                      */
*****************************************************************************/
at minscaling(float minwin_len) {

float minv;

fdef USEINVALES
if    nwin_len > 160) {
  ,   :f("\nERROR: No valid approximation for minv available! (function minscaling)
\n");
  terminate(1);
}
else if (minwin_len >= 60)
  minv = 1.0 / (pow(fabs(log(minwin_len)) / 6, 0.4) 0.035);
else if (minwin_len > 5)
  minv = 1.0 / (pow(fabs(log(minwin_len)) / 6.7, 0.65)+0.102);
else
  minv = 1 / 0.5;

lse
if (minwin_len > 160) {
  printf("\nERROR: No valid approximation for minv available! (function minscaling)
\n");
  terminate(1);
}
else if (minwin_len >= 60)
  minv = 1.0 / (powf(fabsf(logf(minwin_len)) / 6, 0.4)-0.035);
else if (minwin_len > 5)
  minv = 1.0 / (powf(fabsf(logf(minwin_len)) / 6.7, 0.65)+0.102);
else
  minv = 1 / 0.5;


ndif

return(minv);



*****************************************************************************/
      Subroutine init_noise_params_minstat: initialize parameters of noise   */
                         estimation procedure                                */
*****************************************************************************/
id init_noise_params_minstat(Enhance_Params* p)

p->NP->wtr_front_len = 32;
p->NP->hear_thr_rms = 6.0;

ifdef USEDOUBLES
p->NP->resn_thr = 20.0 * log10 (p->NP->hear_thr_rms); /* desired residual abs noise
 el" */
p->NP->GAMAV_TH = 2. * log ( 2. );            /* Upper thid on gamma_av for noise on
 cond. */
lse
  ->NP->resn_thr = 20.0 * log10f (p->NP->hear_thr_rms); /* desired residual abs noise
```

```
 level */
  p->NP->GAMAV_TH = 2. * logf ( 2. );            /* Upper thid on gamma_av for noise o
nly cond. */
#endif

  p->NP->GM_MIN= 0.12;   /* max value for Min. Gain Modif. Factor - GM_min */
  p->NP->nsmth_bias = 1.0 + (1.0 - p->win_ratio ) / 3.0; /* bias factor for initial no
ise estimate */
  p->NP->noise_b_by_nsmth_b = p->noise_bias * p->NP->nsmth_bias;

  p->NP->gammav_thr = 2.0*p->gN;
  p->NP->gammax_thr = 50.0*p->gN;   /* 10 for stationary noise, 18 for babble */

  p->NP->len_minwin = (int) ceil(12.0 / p->win_shift_ratio);

  p->NP->num_minwin = 8;
  p->NP->Dmin=p->NP->len_minwin * p->NP->num_minwin;

  if (p->software_ver == 6)
    p->NP->alpha_N_max = 0.96;
  else
    p->NP->alpha_N_max = 0.94;

  if (p->software_ver >= 8)
    p->NP->Pdecay_num = - 1. / (4. / p->win_shift_ratio);
  else
    p->NP->Pdecay_num = 0.0;


  p->NP->minv = minscaling(p->NP->Dmin);
  p->NP->minv_sub = minscaling(p->NP->len_minwin);
  p->NP->Fvar = (p->NP->Dmin-1)*(p->NP->minv 1);
  p->NP->Fvar_sub = (p->NP->len_minwin-1)*(p->NP->minv_sub 1);
 }

#endif

 /*****************************************************************************/
 /*      Subroutine init_params: initialize parameters of enhancement program   */
 /*****************************************************************************/
 void init_params(Enhance_Params* p, const char* version_name)
 {

   if (!strcmp(version_name,"nsa6"))
      p->software_ver = 6;
   else if (!strcmp(version_name,"nsa7"))
      p->software_ver = 7;
   else if (!strcmp(version_name,"nsa8"))
      p->software_ver = 8;
   else {
     printf("\nERROR: %s is no valid preprocessor version! (init_params)\n\n", version
_name);
     terminate(1);
   };

   p->ENVLP_FLG = 1;           /* Set to 0 for no lower_envelope tracking
                                  Set to 1 for tracking with Spectral Sampling
                                  Set to 2 for tracking with Mean Matching only */

   p->NS_FLG = 0;              /* Set to 1 to update noise_spec - signal present */

   p->fs = 8000;

   /* length of analysis window; if you change this you have to supply a new windows.h
      file or come up with functions to generate the windows during run time */
```

02/05/99
15:59:28

cnh_fun.c

```
p->win_len = 256;
p->win_len_inv = 1.0 /p->win_len;

if (p->software_ver >= 7) {
  p->win_shift = 180;      /* frame advance suitable for MELP coder */
  p->analysis_window = sqrt_tukey;
}

else {
  p->win_shift = 128;      /* standard frame advance */
  p->analysis_window = hann_even;
}

p->overlap_len = p->win_len - p->win_shift;
p->win_ratio = p->win_len/256;
p->win_shift_ratio = 2.0 * p->win_shift / p->win_len;

p->[illegible]_lenf = p->win_len/2 + 1;
p->rate = 8;
p->rate_factor = p->rate;

#ifdef USEDOUBLES
 p->noise_bias = sqrt(2.0);    /* noise spectral bias factor (1.5dB)  */
#else
 p->noise_bias = sqrtf(2.0);   /* noise spectral bias factor (1.5dB)  */
#endif
 p->qN = 1/p->noise_bias;
 p->noise_frames = 1;

 p->alpha_LT = 1.0 - (1.0 / ((int)((1.0*p->fs)/p->win_shift)*1.0))*p->win_ratio;
 p->beta_LT = 1.0 - p->alpha_LT;

 p->qk_max = 1. - 0.001;
 p->qk_min = 0.001;

 if (p->software_ver == 6)
   p->alphak = 0.99 - (p->rate / 16.0 ) * 0.08;
 else
   p->alphak = 0.99 - (p->rate / 16.0 ) * 0.12;

 p->betak = 1. - p->alphak;

 p->alphay = 1. - p->win_ratio;
 p->betay = 1. - p->alphay;

 p->ksi_minq = 0.2;

#ifdef USEDOUBLES
 p->vb = log(1./(1. - 0.5));  /*  'false alarm' prob. for setting thld to find qk's
/
#else
 p->vb = logf(1./(1. - 0.5));  /*  'false alarm' prob. for setting thld to find qk's
*/
#endif
 p->gammaq_thr = (1. + p->ksi_minq)*p->vb*p->gN;

 /* Smoothing factor of hard decision qk-value  */
 p->alphaq = 0.99 - p->win_ratio*0.04*p->win_shift_ratio;
 p->betaq = 1. - p->alphaq;
   /
#ifdef MALAH
 p->NP = malloc(sizeof(Noise_Params_Malah));
 init_noise_params_malah(p);
#else
 p->NP = malloc(sizeof(Noise_Params_Minstat));
  init_noise_params_minstat(p);
#endif

  if (p->NP == NULL)
   {
    printf("\nERROR: malloc request cannot be satisfied! (init_params)\n\n");
    terminate(1);
   };

}


/*************************************************************************/
/*      Subroutine gain_mod: compute gain modification factor based on    */
/*         signal absence probabilities qk                                 */
/*************************************************************************/
Float *gain_mod(Float* GM, Float* qk, Float* ksi, Float* vk, int m)
{

  int i;
  Float temp, ksiq;

   for (i = 0; i < m; i++)
     {
       temp = 1.0 - qk[i];
       ksiq = ksi[i] / temp;

#ifdef USEDOUBLES
       GM[i] = temp / ( temp + (1+ksiq) * qk[i] * exp(-vk[i]) );
#else
       GM[i] = temp / ( temp + (1+ksiq) * qk[i] * expf(-vk[i]) );
#endif
     };

   return (GM);
}


/*************************************************************************/
/*      Subroutine compute_qk: compute the probability of speech absence   */
/*         This uses an approach due to Malah (1996).                      */
/*************************************************************************/
Float *compute_qk(Float *qk,Float *gamaK,Float *ksi,int m)
{

  Float tmp1, temp, ksi_vt;
  int i;

  for ( i = 0; i < m; i++ )
   {
     tmp1 = 2.0*ksi[i];
     ksi_vt =  tmp1 / (1.0+tmp1);
#ifdef USEDOUBLES
     temp = (1. + 2. * ksi[i]) * exp (-ksi_vt * gamaK[i]);
#else
     temp = (1. + 2. * ksi[i]) * expf (-ksi_vt * gamaK[i]);
#endif
     qk[i] = temp / (1. + temp);

   };

 return(qk);

}
/*************************************************************************/
```

```
*      Subroutine compute_qk_new: compute the probability of speech absence   */
*        This uses an harddecision approach due to Malah (ICASSP 1999).       */
******************************************************************************/
loat *compute_qk_new(Float *qk,Float *qla, Float *gamaK,Float GammaQ_TH, Float alpha
 Float betaq, int m)

 int i;

 for ( i = 0; i < m; i++ )
  {
   qla[i] = alphaq*qla[i];

   if (gamaK[i] < GammaQ_TH)
     qla[i] += betaq;

   qk[i] = qla[i];



return(qk);




******************************************************************************/
*      Subroutine gain_log_mmse: compute the gain factor and the auxiliary    */
*        variable vk for the Ephraim&Malah 1985 log spectral estimator.       */
*      Approximation of the exponential integral due to Malah, 1996           */
******************************************************************************/
loat *gain_log_mmse(Float *Gain,Float *vk,Float *qk,Float *ksi,Float *gamaK,int m)


 int i;
 Float temp, ksiq_inv, ksi_vq, eiv;

 for ( i = 0; i < m; i++ )
   {
     temp = 1. - qk[i];
     ksiq_inv = temp / ksi[i];
     ksi_vq = 1. / ( 1. + ksiq_inv );

     vk[i] = ksi_vq * gamaK[i];
     if (vk[i] < 2.220446049250313e-16)
       vk[i] = 2.220446049250313e-16;    /* MATLAB eps */

#ifdef USEDOUBLES
     if (vk[i] < 0.1)
       eiv = -2.31 * log10(vk[i]) - 0.6;
     else
       if (vk[i] > 1)
         eiv = pow( 10, -0.52 * vk[i] - 0.26 );
       else
         eiv = -1.544 * log10(vk[i]) + 0.166;

     Gain[i] = ksi_vq * exp (0.5 * eiv);
#else
     if (vk[i] < 0.1)
       eiv = -2.31 * log10f(vk[i]) - 0.6;
     else
       if (vk[i] > 1)
         eiv = powf( 10,  0.52 * vk[i] - 0.26 );
       else
         eiv =  1.544 * log10f(vk[i]) + 0.166;

     Gain[i] = ksi_vq * expf (0.5 * eiv);
```

```
#endif

     };

   return(Gain);
 }



/*****************************************************************************/
/*      Subroutine ksi_min_adapt: computes the adaptive ksi_min              */
/*****************************************************************************/
Float ksi_min_adapt(int n_flag,Float Ksi_min, Float sn_lt)
{
     Float Ksi_min_new;

        if (n_flag == 1)  /* RM: adaptive modification of ksi limit (10/98) */
          Ksi_min_new = Ksi_min;
        else
          {
#ifdef USEDOUBLES
          Ksi_min_new = Ksi_min*exp(0.65*log(0.5+sn_lt) - 5);
#else
          Ksi_min_new = Ksi_min*expf(0.65*logf(0.5+sn_lt) - 5);
#endif
          if (Ksi_min_new > 0.25)
            Ksi_min_new = 0.25;
          }

        return(Ksi_min_new);
}


/*****************************************************************************/
/*      Subroutine smoothing_win: applies the Parzen  window                 */
/*****************************************************************************/
void smoothing_win ( Float *tempV2, Enhance_Params *P) {
    int i;
    Float *fp;

    fp = tempV2 + P->win_len - 1;

    for ( i = 1; i < P->NP->wtr_front_len; i++, fp-- ) {
        tempV2[i] *= wtr_front[i];
        fp[0] *= wtr_front[i];
    }

    while ( i < P->win_len - P->NP->wtr_front_len + 1 )
        tempV2[i++] = 0.0;
}

#ifdef MALAH
/*****************************************************************************/
/*      Subroutine gain_log_mmse: compute the gain factor and the auxiliary  */
/*        variable vk for the Ephraim&Malah 1985 log spectral estimator.     */
/*      Approximation of the exponential integral due to Malah, 1996         */
/*****************************************************************************/
 void track_envelope(Float YY_av, Enhance_Data *D, Enhance_Params *P)
 {
   Float denv;
   Float envlp_lin;
   Float envlp_alpha;
   Float sum = 0.0;
   int i;
```

```
D->env_flg = 0;

denv = D->envlp - YY_av;
if (denv >= 0.) {    /* drop condition */
  D->envlp = YY_av;
  D->env_drop_flg = 1;
     } else {
         envlp_lin = D->envlp * P->NP->env_rate;
         envlp_alpha = D->envlp * P->NP->alpha_env + P->NP->beta_env * YY_av;
         D->envlp = envlp_lin > envlp_alpha ? envlp_lin : envlp_alpha;

         if (D->env_drop_flg == 1)
             D->env_flg = 1;
         D->env_drop_flg = 0;
     }

     /* conditional update noise-spect (envlp) and related variables */

     if ( D->env_flg == 1 &&
         ( (D->envlp > 2. * D->N_pwr) || (D->envlp < D->N_pwr * 0.5) ) ) {

         if (P->ENVLP_FLG == 2) {       /* matching to mean spectrum only */

             if ( ( D->EY_av < 1.414 * D->YY0_av ) &&   /* +- 1.5dB range */
                 ( D->EY_av > D->YY0_av * 0.707214 ) ) {

                 for ( i = 0; i < P->win_len/2 + 1; i++)  /* sample mean spec */
                     D->lambdaD[i] = P->noise_bias * D->EY[i];

                 D->updn_flg = 3;   /* lock to mean (EY) (indicator) */
                 D->YY_flg = 0;
             }
         } else {

             if (D->envlp > 2. * D->N_pwr) {       /* increase */

                 if ( ( D->EY_av > 1.414 * D->YY0_av ) ||  /* +- 1.5dB range */
                     ( D->EY_av < D->YY0_av * 0.707214 ) ) {

                     for ( i = 0; i < P->win_len/2 + 1; i++ ) {
#i     USEDOUBLES
                         D->lambdaD[i] = sqrt(D->N_pwr / D->envlp) * D->YY_smth[i]

#else

                         D->lambdaD[i] = sqrtf(D->N_pwr / D->envlp) * D->YY_smth[i

#endif

                     }

                     D->YY_flg = 1;
                     D->updn_flg = 2;    /* up to YY_smth (indicator) */

                 } else {

                     for ( i = 0; i < P->win_len/2 + 1; i++ )  /* sample mean spec
*/
                         D->lambdaD[i] = D->EY[i];

                     D->updn_flg = 1;
                     D->YY_flg = 0;
                 }

                 for ( i = 0; i < P->win_len/2 + 1; i++ )   /* mult by bias factor
*/
                         D->lambdaD[i] *= P->noise_bias;

             } else {                    /* D->envlp < D->N_pwr / 2. (drop) */

                 D->updn_flg = -1;    /* 'natural' change of lambdaD (indicator) *
/
                 if (D->YY_flg == 1) {

                     for ( i = 0; i < P->win_len/2 + 1; i++ ) {
#ifdef USEDOUBLES
                         D->lambdaD[i] = sqrt(D->N_pwr / D->envlp) * D->YY_smth[i]
;
#else
                         D->lambdaD[i] = sqrtf(D->N_pwr / D->envlp) * D->YY_smth[i
];
#endif
                     }

                     D->updn_flg = -2;  /* half-way drop (indicator) */
                 }
             }
         }    /* end matching to mean spectrum only check */

         /* update N_pwr0, GM_min and Ksi_min */

         sum = D->lambdaD[0] + D->lambdaD[P->win_len/2];
         for ( i = 1; i < P->win_len/2; i++ )
             sum += D->lambdaD[i] + D->lambdaD[i];
         D->N_pwr0 = sum * P->win_len_inv;

#ifdef USEDOUBLES
         D->n_pwr = 10 * log10( 1 + D->N_pwr0 );
#else
         D->n_pwr = 10 * log10f( 1 + D->N_pwr0 );
#endif

         D->ndiff = D->n_pwr - P->NP->reen_thr;

#ifdef USEDOUBLES
         D->GM_min = pow( 10, -D->ndiff * 0.05 );
#else
         D->GM_min = powf( 10, -D->ndiff * 0.05 );
#endif

         if ( D->GM_min > P->NP->GM_MIN )
             D->GM_min = P->NP->GM_MIN;

         D->Ksi_min = D->GM_min * P->rate_factor;

         if ( D->Ksi_min > P->NP->GM_MIN )
             D->Ksi_min = P->NP->GM_MIN;

         if ( D->GM_min < D->Ksi_min )
             D->GM_min = D->Ksi_min;
     }
}

/*****************************************************************************/
/*      Subroutine update_noise_spect: update the noise power spectral density */
/*        This is a part of Malahe noise estimation procedures.              */
/*****************************************************************************/
void update_noise_spect(float gamma_av, int n_flag, Enhance_Data *D, Enhance_Params *P
)
{
```

```
Float sum, sum2, temp, alphaDn, betaDn, *fp;
int i, count;

sum = D->lambdaD[0] + D->lambdaU[P->win_len/2];
for ( i = 1; i < P->win_len/2; i++ )
    sum += D->lambdaD[i] + D->lambdaD[i];

D->N_pwr0 = sum * P->win_len_inv;

if ( (n_flag == 1) && (D->n_c10 == 0) && (D->n_c01 == 0) ) {  /* NOISE ONLY */

    temp = gamma_av - P->gN;
    betaDn = P->NP->f_n_max * ( temp > 0 ? temp : -temp );

    if ( betaDn > P->NP->f_n_max )
        betaDn = P->NP->f_n_max;
    if ( betaDn < P->NP->f_n_min )
        betaDn = P->NP->f_n_min;
    alphaDn = 1. - betaDn;

    for ( i = 0; i < P->win_len/2 + 1; i++ )
        D->lambdaD[i] = alphaDn * D->lambdaD[i] + P->noise_bias * betaDn * D->YY[i

} else {                                                  /* SIGNAL PRESENT */
    if ( P->NS_FLG ) {

        temp = P->NP->f_s_min;
        sum = 0.0;
        count = 0;

        if ( D->gamaK[0] <= P->NP->gammaxs_thr ) {
            sum += D->gamaK[0];
            count ++;
        }
        if ( D->gamaK[P->win_len/2] <= P->NP->gammaxs_thr ) {
            sum += D->gamaK[P->win_len/2];
            count ++;
        }

        for ( i = 1; i < P->win_len/2; i++ ) {
            if ( D->gamaK[i] <= P->NP->gammaxs_thr ) {
                sum += D->gamaK[i] + D->gamaK[i];
                count += 2;
            }
        }

        sum2 = sum / count - P->gN;

        if ( count > 0 )
            temp = P->NP->f_s_max * ( sum2 > 0 ? sum2 : -sum2 );

        if ( temp > P->NP->f_s_max )
            temp = P->NP->f_s_max;
        if ( temp < P->NP->f_s_min )
            temp = P->NP->f_s_min;

        for ( i = 0; i < P->win_len/2 + 1; i++ )
            if ( D->gamaK[i] <= P->NP->gammaxs_thr )
                D->lambdaD[i] += temp * D->qk[i] * (P->noise_bias * D->YY[i] - D->
ambdaD[i] );
```

```
    if ( P->ENVLP_FLG ) {     /* Smoothing YY */

        if (D->env_drop_flg == 1) {

            fp = D->tempV1;
            for ( i = 0; i < P->win_len/2 + 1; i++, fp += 2 ) {
                fp[0] = D->YY[i];
                fp[1] = 0.0;
            }

            ifftr ( D->tempV2, D->tempV1, &D->fcache );

            smoothing_win ( D->tempV2 , P);

            fftr ( D->tempV1, D->tempV2, &D->fcache );

            fp = D->tempV1;
            for ( i = 0; i < P->win_len/2 + 1; i++, fp += 2 )
                D->YY_smth[i] = fp[0];
        }
    }
}
#endif

#ifdef MINSTAT
/******************************************************************************/
/*      Subroutine smoothed_periodogram: compute short time psd with optimal   */
/*                          smoothing                                          */
/******************************************************************************/
Float* smoothed_periodogram(Enhance_Data *D, Float YY_av, Enhance_Params *P) {

    int i;
    Float smoothed_av, alphacorr_new, alpha_N_min, alpha_N_min_1, alpha_N_min_2, alpha
_num, temp;

    smoothed_av = D->smoothedspect[0]+D->smoothedspect[P->vec_lenf-1]
                + 2*vec_sum(&D->smoothedspect[1],P->vec_lenf-2);
    smoothed_av = smoothed_av * P->win_len_inv;
    alphacorr_new = smoothed_av / YY_av - 1.0;
    alphacorr_new = 1.0 / (1.0 + alphacorr_new * alphacorr_new);
    D->alphacorr = 0.7*D->alphacorr + 0.3 * (alphacorr_new > 0.7 ? alphacorr_new : 0.7
);

#ifdef USEDOUBLES
    alpha_N_min_1 = pow(D->SN_LT,(P->NP->Pdecay_num));
#else
    alpha_N_min_1 = powf(D->SN_LT,(P->NP->Pdecay_num));
#endif

    alpha_N_min_2 = (0.3 < alpha_N_min_1 ? 0.3 : alpha_N_min_1);

    alpha_N_min = (alpha_N_min_2 < 0.05 ? 0.05 : alpha_N_min_2);
    alpha_num = P->NP->alpha_N_max * D->alphacorr;

    for (i = 0; i < P->vec_lenf; i++) {
      temp = (D->smoothedspect[i]/D->noisespect[i] 1);
      D->alpha_var[i] = alpha_num / (1+temp * temp);
      D->alpha_var[i] = (D->alpha_var[i] < alpha_N_min ? alpha_N_min : D->alpha_var[i]
);
      D->smoothedspect[i] = D->alpha_var[i]* D->smoothedspect[i] + (1-D->alpha_var[i])
 * D->YY[i];

    }
```

02/05/99
15:59:28

enh_fun.c.

```
  return(D->smoothedspect);

*****************************************************************************/
*       Subroutine normalized_variance: compute variance of smoothed          */
*                       periodogram, normalize it, and use it to compute a    */
*                       biased smoothed periodogram                           */
*****************************************************************************/
loat* bias_compensation(Enhance_Data *D, Enhance_Params *P) {

 int i;
 Float tmp1, tmp2, beta_var, var_sp, var_rel_av_sqrt;

 for (i = 0; i < P->vec_lenf; i++) {
   tmp1 = D->alpha_var[i]*D->alpha_var[i];
   beta_var = (tmp1 > 0.8 ? 0.8 : tmp1);
   tmp2 = (1-beta_var)* D->smoothedspect[i];

      ar_sp_av[i] = beta_var * D->var_sp_av[i] + tmp2;
   D->var_sp_2[i] =  beta_var * D->var_sp_2[i] + tmp2 * D->smoothedspect[i];
   var_sp = D->var_sp_2[i] - D->var_sp_av[i] * D->var_sp_av[i];
   tmp1 = var_sp / (D->noisespect[i] * D->noisespect[i]);
   D->var_rel[i] = (tmp1 > 0.5 ? 0.5 : tmp1);
 };

 D->var_rel_av = (D->var_rel[0]+D->var_rel[P->vec_lenf-1]
           + 2 * vec_sum(&D->var_rel[1],P->vec_lenf-2)) * P->win_len_inv;
 D->var_rel_av = (D->var_rel_av < 0 ? 0 : D->var_rel_av);

ifdef USEDOUBLES
 var_rel_av_sqrt =  1.0 + 1.5*sqrt(D->var_rel_av);
else
 var_rel_av_sqrt =  1.0 + 1.5*sqrt((D->var_rel_av);
endif

 tmp1 = var_rel_av_sqrt*P->NP->fvar;
 tmp2 = var_rel_av_sqrt*P->NP->fvar_sub;

 for (i = 0; i < P->vec_lenf; i++) {
   D->biased_smoothedspect[i] = (var_rel_av_sqrt + D->var_rel[i]
                               * tmp1 / (P->NP->minv-D->var_rel[i])) * D->smootheds
ect[i];
      biased_smoothedspect_sub[i] = (var_rel_av_sqrt + D->var_rel[i]
                               * tmp2 / (P->NP->minv_sub-D->var_rel[i])) * D->smoot
edspect[i];
 };

 return(D->biased_smoothedspect);



*****************************************************************************/
*       Subroutine noise_slope: compute maximum of the permitted increase of  */
*          the noise estimate as a function of the mean signal variance      */
*****************************************************************************/
loat noise_slope(Enhance_Data *D, Enhance_Params *P) {

 Float noise_slope_max;

 if (D->var_rel_av > 0.18)
   noise_slope_max = 1.2;
 else if ((D->var_rel_av < 0.03) || (D->I < 50))
   noise_slope_max  = 8;
 else if (D->var_rel_av < 0.05)
   noise_slope_max  = 4;
 else if (D->var_rel_av < 0.06)
   noise_slope_max  = 2;
 else
   noise_slope_max  = 1.2;

 if (P->software_ver >= 7 )
   noise_slope_max *= 1.1;

 return(noise_slope_max);
}

/*****************************************************************************/
/*      Subroutine min_search: find minimum of psd's in circular buffer      */
/*****************************************************************************/
Float* min_search(Enhance_Data *D, Enhance_Params *P) {

    int i,k;
    Float noise_slope_max;

    if (D->minspec_counter >= P->NP->len_minwin) {

      noise_slope_max = noise_slope(D,P);

      for (i = 0; i < P->vec_lenf; i++) {
        if (D->biased_smoothedspect[i] < D->act_min[i]) {
          D->act_min[i] = D->biased_smoothedspect[i];         /* update minimum */
          D->act_min_sub[i] = D->biased_smoothedspect_sub[i];
          D->localflag[i] = 0;
        };
        D->circb[D->circb_indx][i] = D->act_min[i];     /* write new minimum into ring
buffer */
        D->circb_indx = (((D->circb_indx +1) %
                   P->NP->num_minwin) == 0) ? 0 : (D->circb_indx +1)); /* increment r
b pointer */

        D->circb_min[i] = D->circb[0][i];             /* find minimum of ring buffer */
        for (k = 1; k<P->NP->num_minwin; k++) {
            D->circb_min[i] = D->circb[k][i] < D->circb_min[i] ? D->circb[k][i] : D->c
ircb_min[i];
        };

        /* rapid update in case of local minima which do not deviate more than noise_s
lope_max
           from the current minima */
        if (D->localflag[i] && (D->act_min_sub[i] > D->circb_min[i]) &&
           (D->act_min_sub[i] < noise_slope_max * D->circb_min[i])) {
          D->circb_min[i] = D->act_min_sub[i]; /* (D->circb_min[i] < D->act_min_sub[i]
 ?
                              D->act_min_sub[i] : D->circb_min[i]); */

          /* propagate new rapid update minimum into ring buffer */
          for (k = 0; k<P->NP->num_minwin; k++)
                  D->circb[k][i]=D->circb_min[i];
        };

        D->act_min[i] = FLT_MAX;           /* initialize minimum with largest possible
value */
        D->localflag[i] = 0;               /* reset local minimum indicator */
      };
      D->minspec_counter = 1;

    }
    else {
      if (D->minspec_counter > 1) {
        for (i = 0; i < P->vec_lenf; i++) {
```

```
        if (D->biased_smoothedspect[i] < D->act_min[i]) {
          D->act_min[i] = D->biased_smoothedspect[i];        /* update minimum */
          D->act_min_sub[i] = D->biased_smoothedspect_sub[i];
          D->localflag[i] = 1;
        };
        D->noisespect[i] = (D->act_min_sub[i] < D->circb_min[i] ? D->act_min_sub[i]
D->circb_min[i]);
        D->circb_min[i] = D->noisespect[i];
        D->lambdaD[i] = P->noise_bias * D->noisespect[i];
      };
    }
    else {
      for (i = 0; i < P->vec_len; i++) {
        if (D->biased_smoothedspect[i] < D->act_min[i]) {
          D->act_min[i] = D->biased_smoothedspect[i];        /* update minimum */
          D->act_min_sub[i] = D->biased_smoothedspect_sub[i];
        };
      };

    };

    (D->minspec_counter)++;
  };

  return(D->lambdaD);

~ndif

*****************************************************************************/
*      Subroutine enh_init: initialization of variables for the enhancement   */
*****************************************************************************/
~id enh_init( Enhance_Data *D, Enhance_Params *P)

  int i, j;
  Float sum = 0.0;

  Float *fp, *fp2;

  /* allocate arrays */
  D->qk = CALLOC_FLOAT(P->vec_len);
  D->qla = CALLOC_FLOAT(P->vec_len);
  * ~lambdaD = CALLOC_FLOAT(P->vec_len);

      .Y0 = CALLOC_FLOAT(P->vec_len);
  D->Agal = CALLOC_FLOAT(P->vec_len);

  D->tempV1 = CALLOC_FLOAT(P->win_len+2);
  D->tempV2 = CALLOC_FLOAT(P->win_len);
  D->tempV3 = CALLOC_FLOAT(P->win_len+2);

  D->analy = CALLOC_FLOAT(P->win_len);
  D->Y = CALLOC_FLOAT(P->win_len+2);
  D->YY = CALLOC_FLOAT(P->win_len/2 + 1);
  D->Ymag = CALLOC_FLOAT(P->win_len/2 + 1);
  D->gamaK = CALLOC_FLOAT(P->win_len/2 + 1);
  D->ksi = CALLOC_FLOAT(P->win_len/2 + 1);

  D->Gain = CALLOC_FLOAT(P->win_len/2 + 1);
  D->GM = CALLOC_FLOAT(P->win_len/2 + 1);
  D->GainD = CALLOC_FLOAT(P->win_len/2 + 1);

  D->vk = CALLOC_FLOAT(P->win_len/2 + 1);

  D->ygal = CALLOC_FLOAT(P->win_len+2);
```

```
    /* set up FFT cache */

    fftrinit( &D->fcache, 8 );

    /* initialize noise spectrum */
    for (j = 0; j < P->win_len+2; j++)
        D->tempV3[j] = 0.0;

    fp2 = D->initial_noise;
    for (j = 0; j < P->noise_frames; j++) {

        for (i = 0; i < P->win_len; i++) {
            D->tempV2[i] = P->analysis_window[i] * fp2[i];
        }

        fftr ( D->tempV1, D->tempV2, &D->fcache );

        /* get rough estimate of lambdaD  */

        D->tempV1[0] = D->tempV1[0]*D->tempV1[0] * P->win_len_inv;
        D->tempV1[1] = 0.0;
        D->tempV1[P->win_len] = D->tempV1[P->win_len]*D->tempV1[P->win_len] * P->win_l
en_inv;
        D->tempV1[P->win_len+1] = 0.0;

        fp = D->tempV1 + 2;
        for ( i = 2; i < P->win_len/2 + 1; i++, fp += 2 ) {
            fp[0] = ( fp[0] * fp[0] + fp[1] * fp[1] ) * P->win_len_inv;
            fp[1] = 0.0;
        }

        for ( i = 0; i < P->win_len+2; i++ )
            D->tempV3[i] += D->tempV1[i];

        fp2 += P->win_shift;
    }

    for ( j = 0; j < P->win_len+2; j++ )
        D->tempV1[j] = D->tempV3[j] / (Float) P->noise_frames;

    /* then smooth nn to get lambdaD if we didn't average */

    if ( P->noise_frames == 1 ) {

        ifftr ( D->tempV2, D->tempV1, &D->fcache );

        smoothing_win ( D->tempV2 , P);

        fftr ( D->tempV1, D->tempV2, &D->fcache );

        D->lambdaD[0] = P->NP->noise_b_by_nmth_b * D->tempV1[0] + 0.001;
        D->lambdaD[P->win_len/2] = P->NP->noise_b_by_nmth_b * D->tempV1[P->win_len] +
0.001;

        sum = D->lambdaD[0] + D->lambdaD[P->win_len/2];

        fp = D->tempV1 + 2;
        for ( i = 1; i < P->win_len/2; i++, fp += 2 ) {
            D->lambdaD[i] = P->NP->noise_b_by_nmth_b * fp[0] + 0.001;
            sum += D->lambdaD[i] + D->lambdaD[i];
        }

    } else {

        D->lambdaD[0] = P->noise_bias * D->tempV1[0] + 0.001;
    D->lambdaD[P->win_len/2] = P->noise_bias * D->tempV1[P->win_len] + 0.001;

    sum = D->lambdaD[0] + D->lambdaD[P->win_len/2];

    fp = D->tempV1 + 2;
    for ( i = 1; i < P->win_len/2; i++, fp += 2 ) {
        D->lambdaD[i] = P->noise_bias * fp[0] + 0.001;
        sum += D->lambdaD[i] + D->lambdaD[i];
    }
  }

  D->N_pwr= sum * P->win_len_inv;

  for (i = 0; i < P->win_len/2 + 1; i++) {
      D->YY0[i] = D->lambdaD[i];
      D->Agal[i] = 0.0;
  }

  v..._fill(D->qla, 0.5, P->vec_len);

.fdef MALAH    /* Initializations for Malah's noise estimator */

  D->YY_smth = CALLOC_FLOAT(P->vec_len);
  D->EY = CALLOC_FLOAT(P->vec_len);

  for (i = 0; i < P->win_len/2 + 1; i++)
      D->YY_smth[i] = D->lambdaD[i];

  /* Set GM_min and Ksi_min */

  D->N_pwr0 = D->N_pwr;

.fdef USEDOUBLES
  D->n_pwr = 10 * log10( 1 + D->N_pwr );
  D->ndiff = D->n_pwr - P->NP->rean_thr;
  D->GM_min = pow(10, -D->ndiff / 20.0);
lse
  D->n_pwr = 10 * log10f( 1 + D->N_pwr );
  D->ndiff = D->n_pwr - P->NP->rean_thr;
  D->GM_min = powf(10, -D->ndiff / 20.0);
ndif

      D->GM_min > P->NP->GM_MIN)
      D->GM_min = P->NP->GM_MIN;

  D->Ksi_min = D->GM_min * P->rate_factor;
  if (D->Ksi_min < 0.1)
      D->Ksi_min = 0.1;
  if (D->Ksi_min > P->NP->GM_MIN)
      D->Ksi_min = P->NP->GM_MIN;
  if (D->GM_min < D->Ksi_min)
      D->GM_min = D->Ksi_min;

  D->G0_min = D->GM_min;

  D->envlp = D->N_pwr;
  D->env_flg = 0;
  D->env_drop_flg = 0;

  ,
  D->updn_flg = 0;
  D->YY_flg = 0;

lse      /* Initializations for the minimum statistics noise estimator */
  minstat_init(D,P);
#endif

    vec_mult(D->tempV1,P->analysis_window,P->analysis_window,P->win_len);

    /* compute initial long term SNR; speech signal power depends on the window;
       the Hanning window is used as a reference here with a squared norm of 96 */
    D->SN_LT = 1.e+6 / D->N_pwr * vec_sum(D->tempV1, P->win_len) / (96.0*P->win_ratio)
;

    D->SN_LT0 = D->SN_LT;
    D->YY_LT = 0.;
    D->Ksi_min_var = D->Ksi_min;

}

#ifdef MINSTAT
/***************************************************************************/
/*      Subroutine minstat_init: Initialization of variables for minimum   */
/*                               statistics noise estimation               */
/***************************************************************************/
void minstat_init(Enhance_Data *D, Enhance_Params *P)
{
    /* Initialize Minimum Statistics Noise Estimator */
    int i,k;

    D->smoothedspect = CALLOC_FLOAT(P->vec_len);
    D->biased_smoothedspect = CALLOC_FLOAT(P->vec_len);
    D->biased_smoothedspect_sub = CALLOC_FLOAT(P->vec_len);
    D->circb_min = CALLOC_FLOAT(P->vec_len);
    D->act_min = CALLOC_FLOAT(P->vec_len);
    D->act_min_sub = CALLOC_FLOAT(P->vec_len);
    D->noisespect = CALLOC_FLOAT(P->vec_len);
    D->alpha_var = CALLOC_FLOAT(P->vec_len);

    D->circb = CALLOC_FLOATP(P->NP->num_minwin);    /* ring buffer */

    for (i=0;i< P->NP->num_minwin;i++) {
      D->circb[i] = CALLOC_FLOAT(P->vec_len);
      for (k = 0; k < P->vec_len; k++) {
        D->circb[i][k] = D->lambdaD[k] * P->gN;
      };
    };
    D->circb_indx = 0;    /* ring buffer pointer */

    D->localflag = CALLOC_SHORT(P->vec_len);
    D->minspec_counter = P->NP->len_minwin;

    D->var_sp_av = CALLOC_FLOAT(P->vec_len);
    D->var_sp_2 = CALLOC_FLOAT(P->vec_len);
    D->var_rel = CALLOC_FLOAT(P->vec_len);


    for (k = 0; k < P->vec_len; k++) {
      D->smoothedspect[k] = D->lambdaD[k] * P->gN;
      D->act_min[k] = D->lambdaD[k]* P->gN;
      D->act_min_sub[k] = D->lambdaD[k]* P->gN;
      D->noisespect[k] = D->lambdaD[k]* P->gN;
      D->circb_min[k] = D->lambdaD[k]* P->gN;

      D->var_sp_av[k] = 1.22474487139159*D->noisespect[k];  /* sqrt(3/2) */
      D->var_sp_2[k] = 2*D->noisespect[k] * D->noisespect[k];
    };

    D->alphacorr=0.9;
  /* Set GM_min and Kei_min */
(def USEDOUBLES
  D->n_pwr = 10 * log10( 1 + D->N_pwr );
  D->ndiff = D->n_pwr - P->NP->reen_thr;
  D->GM_min = pow(10, -D->ndiff / 20.0);
lse
  D->n_pwr = 10 * log10f( 1 + D->N_pwr );
  D->ndiff = D->n_pwr - P->NP->reen_thr;
  D->GM_min = powf(10, -D->ndiff / 20.0);
ndif

  if (D->GM_min > P->NP->GM_MIN)
      D->GM_min = P->NP->GM_MIN;

  D->Kei_min = D->GM_min * P->rate_factor;
  /* (D->Kei_min < 0.1)
      D->Kei_min = 0.1;
  i. (D->Kei_min > P->NP->GM_MIN)
      D->Kei_min = P->NP->GM_MIN;
  if (D->GM_min < D->Kei_min)
      D->GM_min = D->Kei_min;


*****************************************************************************/
      Subroutine minstat_terminate: Terminate execution of noise minimum    */
                   statistics noise estimator                               */
*****************************************************************************/
id minstat_terminate(Enhance_Data *D, Enhance_Params *P)

 int i;

 free(D->smoothedspect);
 free(D->biased_smoothedspect);
 free(D->biased_smoothedspect_sub);
 free(D->circb_min);
 free(D->act_min);
 free(D->act_min_sub);
 f[illegible] D->noisespect);

 for (i=0; i<P->NP->num_minwin;i++) {
   free(D->circb[i]);
 };

 free(D->circb);    /* ring buffer */
 free(D->localflag);
 free(D->var_sp_av);
 free(D->var_sp_2);
 free(D->var_rel);

ndif

*****************************************************************************/
*       Subroutine enh_terminate: Terminate execution of program            */
*****************************************************************************/
id enh_terminate(Enhance_Data *D, Enhance_Params *P)


fdef MALAN
 P;
 free(D->YY_smth);
 free(D->EY);
```

```
#else
  minstat_terminate(D, P);
#endif

  free(P->NP);

  free(D->qk);
  free(D->qla);
  free(D->lambdaD);
  free(D->YY0);
  free(D->Agal);
  free(D->tempV1);
  free(D->tempV2);
  free(D->tempV3);
  free(D->analy);
  free(D->Y);
  free(D->YY);
  free(D->Ymag);
  free(D->gamaK);
  free(D->kai);
  free(D->Gain);
  free(D->GM);
  free(D->GainD);
  free(D->vk);
  free(D->ygal);
}

/*****************************************************************************/
/*        Subroutine process_frame: Enhance a given frame of speech          */
/*****************************************************************************/
void process_frame(Float *inspeech, Float *outspeech, Enhance_Data *D,Enhance_Params *
P)
{
    int i, n_flag;
    Float sum = 0.0;

    Float *fp, *fp2;

    Float YY_av, gamma_av, gamma_max;

    D->I++;

    /* analysis window */
    vec_mult(D->analy,P->analysis_window,inspeech,P->win_len);

    /* into frequency domain - D->Y, D->YD->Y, YY_av, real and imaginary parts are int
erleaved */
    fftr( D->Y, D->analy, &D->fcache );

    D->YY[0] = D->Y[0] * D->Y[0];
    D->YY[P->vec_lenf-1] = D->Y[P->win_len] * D->Y[P->win_len];

    vec_mag2(&D->YY[1],&D->Y[2],P->vec_lenf-2);
    vec_sqrt(D->Ymag,D->YY,P->vec_lenf);

    vec_mult_const(D->YY,D->YY,P->win_len_inv,P->vec_lenf);
    vec_add_const(D->YY,D->YY,0.001,P->vec_lenf);

    if (P->alphay != 0.)
        for ( i = 0; i < P->win_len/2 + 1; i++)
            D->YY[i] = P->alphay * D->YY0[i] + P->betay * D->YY[i];

    sum = 2*vec_sum(&D->YY[1],P->vec_lenf-2)+D->YY[0]+D->YY[P->vec_lenf-1];
    YY_av = sum * P->win_len_inv;
```

02/05/99
15:59:28

enh_fun.c.

```
  /* computation of lower_envelope and setting env flags */
  /* if (P->ENVLP_FLG)
     track_envelope(YY_av, D, P); */

ifdef MINSTAT

  /* compute smoothed short time periodogram */
  smoothed_periodogram(D, YY_av, P);

  /* compute inverse bias and multiply short time periodogram with inverse bias */
  bias_compensation(D,P);

  /* determine unbiased noise psd estimate by minimum search */
  min_search(D,P);
lse
  vec_fill(D->lambdaD,1000.0,P->vec_len); /* !!!!!!!!!!!!!!!!!!!!! */
nd

  /* compute 'gammas' */
  vec_div(D->gamaK,D->YY,D->lambdaD,P->vec_len);

  gamma_max = vec_max(D->gamaK,P->vec_len);
  sum = D->gamaK[0] + D->gamaK[P->vec_len-1] + 2 * vec_sum(&D->gamaK[1],P->vec_len
-1);

  gamma_av = sum * P->win_len_inv;

  /* determine signal presence */
  n_flag = 0;                               /* default flag - signal present */

  if ((gamma_max < P->NP->gammax_thr) && (gamma_av < P->NP->gammav_thr))
    {
     n_flag = 1;            /* noise-only */
     if (YY_av > D->N_pwr * P->NP->gammav_thr * 2.)
           n_flag = 0;            /* overiding if frame SNR > 3dB (9/98) */
    }


  if ( D->I == 1 ) {
     /* Initial estimation of apriori SNR and Gain */
     n_flag = 1;

     for ( i = 0; i < P->vec_len; i++ ) {
         D->ksi[i] = D->Ksi_min;
         D->qk[i] = P->qk_max;
         D->Gain[i] = D->GM_min;
         D->GM[i] = D->GM_min;
         D->GainD[i] = D->GM_min;
         D->Agal[i] = D->Ymag[i] * D->GM_min;
     }
  } else {          /* D->I > 1 */

     /* estimation of apriori SNR */
     for ( i = 0; i < P->vec_len; i++ ) {
       D->ksi[i] = P->alphak * D->Agal[i] * D->Agal[i] * P->win_len_inv / D->lambda
[i] +
           + P->betak * ( (D->gamaK[i] > P->gN) ? D->gamaK[i]-P->gN : 0 );
     }

     D->Ksi_min_var = 0.9*D->Ksi_min_var + 0.1*ksi_min_adapt(n_flag,D->Ksi_min, D->
N_LT);
     vec_limit_bottom(D->ksi,D->ksi,D->Ksi_min_var,P->vec_len);

     /* estimation of k-th component 'signal absence' prob and gain */
     vec_fill(D->qk, P->qk_max, P->vec_len);   /* default value for qk's % (9/98)
*/

        if (n_flag == 0)
          {                                  /* SIGNAL PRESENT */
          /* computation of the long term SNR */
          if (gamma_av > P->NP->gammav_thr)
            {
              D->YY_LT = D->YY_LT*P->alpha_LT + P->beta_LT*YY_av;
              D->SN_LT = (D->YY_LT/D->N_pwr)  - 1;      /* Long-term S/N  */

              if (D->SN_LT < 0)
                D->SN_LT = D->SN_LT0;

              D->SN_LT0 = D->SN_LT;
          };
/* printf("%d \t %10.10f\n",D->I, D->SN_LT); */
          /* Estimating qk's using Hard-decision Approach   (7/98) */
          compute_qk_new(D->qk,D->qla, D->gamaK,P->gammaq_thr, P->alphaq, P->betaq, P-
>vec_len);

          vec_limit_top(D->qk,D->qk,P->qk_max,P->vec_len);
          vec_limit_bottom(D->qk,D->qk,P->qk_min,P->vec_len);

          }; /* if (n_flag == 0) */


sum=D->qk[0] + D->qk[P->vec_len-1];
for (i=1;i<P->vec_len-1; i++ ) {sum = sum+2* D->qk[i];}
/* printf("%d \t %10.10f\n",D->I, sum);
 vec_fill(D->qk,sum/P->win_len,P->vec_len); */

          gain_log_mmag(D->Gain,D->vk,D->qk,D->ksi,D->gamaK,P->vec_len);   /* o.k.  R
.M. 29/1/99 */
          vec_limit_top(D->Gain,D->Gain,1.0,P->vec_len);        /* limit gain to 1 */

          gain_mod(D->GM,D->qk,D->ksi,D->vk,P->vec_len);                  /* o.k.
R.M. 29/1/99 */
          vec_limit_bottom(D->GM,D->GM,D->GM_min, P->vec_len);  /* limit lowest GM va
lue */

          vec_mult(D->GainD,D->Gain,D->GM,P->vec_len);              /* modified gain *
/

          vec_mult(D->Agal,D->GainD,D->Ymag, P->vec_len);

     };   /* D->I > 1 */

     /* enhanced signal in frequency domain */
     /*   (implement ygal = GainD .* Y)   */

     fp = D->ygal;
     fp2 = D->Y;
     for ( i = 0; i < P->win_len/2+1; i++, fp += 2, fp2 += 2 ) {
       fp[0] =  fp2[0] * D->GainD[i];
       fp[1] =  fp2[1] * D->GainD[i];
     }

     /* transformation to time domain */
     ifftr ( outspeech, D->ygal, &D->fcache );

     if (P->software_ver >= 7)
       vec_mult(outspeech,outspeech,P->analysis_window,P->win_len);

     /* update_noise_spect(gamma_av, n_flag, D, P); */
```

12

15:59:28

cnh_fun.c

```
/* Misc. updates */
D->YY0[0] = D->YY[0];
D->YY0[P->win_len/2] = D->YY[P->win_len/2];

sum = D->lambdaD[0] * D->lambdaD[P->win_len/2];

for ( i = 1; i < P->vec_lenf-1; i++ ) {
    D->YY0[i] = D->YY[i];
    sum += D->lambdaD[i] * D->lambdaD[i];
}

D->N_pwr = sum * P->win_len_inv;
```

02/05/99
15:59:40

enh_fun.h.

```
(ndef __enh_fun__
·fine __enh_fun__
 ----------------------------------------------------------------------
 enh_fun.h - Speech Enhancement Functions

 Author: Reiner Martin, AT&T Labs-Research

 Last Update: $Id:$

 ----------------------------------------------------------------------
/

nclude "globals.h"
nclude "enhance.h"

id init_params(Enhance_Params* P, const char* version_name);

f    ALAM
u    .t_noise_params_malah(Enhance_Params* p);
id track_envelope(Float YY_av, Enhance_Data *D, Enhance_Params *P);
id update_noise_spect(Float gamma_av, int n_flag, Enhance_Data *D, Enhance_Params *P

lse
at* smoothed_periodogram(Enhance_Data *D, Float YY_av, Enhance_Params *P);
id minstat_init(Enhance_Data *D, Enhance_Params *P);
id minstat_terminate(Enhance_Data *d, Enhance_Params *p);
at* min_search(Enhance_Data *D, Enhance_Params *P);
oat minscaling(Float minwin_len);
oat noise_slope(Enhance_Data *D, Enhance_Params *P);
at* bias_compensation(Enhance_Data *D, Enhance_Params *P);
ndif

rt* CALLOC_SHORT(int num_samples);

oat* CALLOC_FLOAT(int num_samples);

oat** CALLOC_FLOATP(int num_samples);

id terminate(int error_num);

oat *gain_mod(Float* GM, Float* qk, Float* ksi, Float* vk, int m);

l.     ompute_qk(Float *qk,Float *gamaK,Float *ksi,int m);

oat *compute_qk_new(Float *qk,Float *qla, Float *gamaK,Float GammaQ_TH, Float alphaq
Float betaq, int m);

oat *gain_log_mmse(Float *Gain,Float *vk,Float *qk,Float *ksi,Float *gamaK,int m);

oat ksi_min_adapt(int n_flag,Float Ksi_min, Float sn_lt);

id enh_init(Enhance_Data *d, Enhance_Params *p);

id enh_terminate(Enhance_Data *d, Enhance_Params *p);

d process_frame(Float inspeech[], Float outspeech[] , Enhance_Data *d, Enhance_Para
: *p);

ndif
```

```
fndef __vect_fun__
efine __vect_fun__
 ....................................................
 vect_fun.h - Functions for MATLAB - like vector operations

 Author: Rainer Martin, AT&T Labs-Research

 Last Update: $Id:$
 ....................................................
/

nclude "globals.h"

id float_to_short(Float input[], short output[], int num_samples);

oat *vec_copy(Float *vec1,Float *vec2,int m);

     ec_accu(Float *vec1,Float *vec2,int m);

oat *vec_add(Float *vec1,Float *vec2,Float *vec3, int m);

oat *vec_mult(Float *vec1,Float *vec2,Float *vec3, int m);

oat *vec_mult_const(Float *vec, Float *vec2, Float c, int m);

oat *vec_div(Float *vec1,Float *vec2,Float *vec3, int m);

oat *vec_inv(Float *vec1,Float *vec2, int m);

oat *vec_mag2(Float *YY,Float *Y,int m);

oat vec_sum(Float *vec, int m);

oat *vec_add_const(Float *vec1,Float *vec2, Float c, int m);

oat vec_max(Float *vec, int m);

oat vec_min(Float *vec, int m);

oat *vec_sqrt(Float *vec1, Float *vec2, int m);

o    vec_limit_bottom(Float *vec1, Float *vec2, Float c, int m);

oat *vec_limit_top(Float *vec1, Float *vec2, Float c, int m);

oat *vec_fill(Float *vec,Float c, int m);

 ndif
```

02/05/99
16:00:03

vect_fun.c.

```
nclude "vect_fun.h"

    ..............................................................
  vect_fun.c - Functions for MATLAB - like vector operations

  Author: Rainer Martin, AT&T Labs-Research

  Last Update: $Id:$

    . ............................ ..............................
 */

*************************************************************************/
      Subroutine float_to_short: round float samples to short samples      */
 ************************************************************************/
d    at_to_short(Float input[], short output[], int num_samples)

   int i;

   for (i = 0; i < num_samples; i++ ) {
     if (input[i] > 32767.)
       output[i] = 32767;
     else if (input[i] < -32768.)
       output[i] = -32768;
     else if (input[i] >= 0.)
       output[i] = (short) (input[i] + .5);
     else
       output[i] = (short) (input[i] - .5);
   }


*************************************************************************/
    Subroutine vec_copy: copy vector vec2 of float samples into vector vec1   */
*************************************************************************/
loat *vec_copy(Float *vec1,Float *vec2,int m)

   int i;

       i=0; i < m; i++)
       vec1[i] = vec2[i];
   return(vec1);


*************************************************************************/
    Subroutine vec_accu: add m samples of vec2 to vec1                        */
*************************************************************************/
loat *vec_accu(Float *vec1,Float *vec2,int m)

   int i;

   for(i=0; i < m; i++)
       vec1[i] += vec2[i];
   return(vec1);


*************************************************************************/
    Subroutine vec_add: add m samples of vec2 to vec3, store result in vec1   */
*************************************************************************/
loat *vec_add(Float *vec1,Float *vec2,Float *vec3, int m)

   int i;
```

```
    for(i=0; i < m; i++)
        vec1[i] = vec2[i]+vec3[i];
    return(vec1);
}

/*************************************************************************/
/*   Subroutine vec_mult: multiply m samples: vec1[i] = vec2[i] * vec3[i]      */
/*************************************************************************/
Float *vec_mult(Float *vec1,Float *vec2,Float *vec3, int m)
{
    int i;

    for(i=0; i < m; i++)
        vec1[i] = vec2[i]*vec3[i];
    return(vec1);
}

/*************************************************************************/
/*   Subroutine vec_div: divide m samples: vec1[i] = vec2[i] /  vec3[i]        */
/*************************************************************************/
Float *vec_div(Float *vec1,Float *vec2,Float *vec3, int m)
{
    int i;

    for(i=0; i < m; i++)
        vec1[i] = vec2[i]/vec3[i];

    return(vec1);
}

/*************************************************************************/
/*   Subroutine vec_inv: inverse m samples: vec1[i] = 1/ vec2[i]               */
/*************************************************************************/
Float *vec_inv(Float *vec1,Float *vec2, int m)
{
    int i;

    for(i=0; i < m; i++)
        vec1[i] = 1/vec2[i];
    return(vec1);
}

/*************************************************************************/
/*    Subroutine vec_mag2: YY is magnitude squared of vector Y.                */
/*    Y contains real and imaginary part interleaved, starting with real part. */
/*************************************************************************/
Float* vec_mag2(Float *YY,Float *Y,int m)
{

  int i;
  Float *fp;

  fp = Y;

  for (i = 0; i < m; i++ , fp += 2)
    YY[i] = fp[0] * fp[0] + fp[1] * fp[1];

  return(YY);
}


/*************************************************************************/
/*    Subroutine vec_sum: computes the sum of vector components.               */
/*************************************************************************/
Float vec_sum(Float *vec, int m)
Float tmp;
int i;

tmp = 0;

for (i = 0; i < m; i++ )
  tmp += vec[i];

return(tmp);



****************************************************************************/
   Subroutine vec_mult_const: multiply m samples with constant:          */
          vec1[i] = vec2[i] * c                                          */
****************************************************************************/
     ec_mult_const(Float *vec1,Float *vec2,Float c, int m)

  int i;

  for(i=0; i < m; i++)
      vec1[i] = vec2[i] * c;

  return(vec1);


****************************************************************************/
   Subroutine vec_add_const: add constant c to m samples:                */
          vec1[i] = vec2[i] + c                                          */
****************************************************************************/
at *vec_add_const(Float *vec1,Float *vec2,Float c, int m)

  int i;

  for(i=0; i < m; i++)
      vec1[i] = vec2[i] + c;

  return(vec1);


...  ***********************************************************************/
     routine vec_sqrt : compute sqrt of vec2: vec1[i] = sqrt(vec2[i])     */
...  ***********************************************************************/
at *vec_sqrt(Float *vec1, Float *vec2, int m)

  int i;

  for(i=0; i < m; i++)
ifdef USEDOUBLES
     vec1[i] = sqrt(vec2[i]);
lse
     vec1[i] = sqrtf(vec2[i]);
ndif

  return(vec1);



****************************************************************************/
   Subroutine vec_max: computes the maximum of m vector components.      */
****************************************************************************/
 at vec_max(Float *vec, int m)

Float tmp;
```

```
  int i;

  tmp = vec[0];

  for (i = 1; i < m; i++ )
    if (vec[i] > tmp)
       tmp = vec[i];

  return(tmp);
}

/****************************************************************************/
/*    Subroutine vec_sum: computes the minimum of vector components.          */
/****************************************************************************/
Float vec_min(Float *vec, int m)
{
  Float tmp;
  int i;

  tmp = vec[0];

  for (i = 1; i < m; i++ )
    if (vec[i] < tmp)
      tmp = vec[i];

  return(tmp);
}

/****************************************************************************/
/*    Subroutine vec_limit_bottom: compare vec2[i] with a constant c and take */
/*     maximum.                                                              */
/****************************************************************************/
Float *vec_limit_bottom(Float *vec1, Float *vec2, Float c, int m)
{
  int i;

  for (i = 0; i < m; i++ )
    vec1[i] = (vec2[i] < c) ? c : vec2[i];

  return(vec1);
}

/****************************************************************************/
/*    Subroutine vec_limit_top: compare vec[i] with a constant c and take    */
/*     minimum.                                                              */
/****************************************************************************/
Float *vec_limit_top(Float *vec1, Float *vec2, Float c, int m)
{
  int i;

  for (i = 0; i < m; i++ )
        vec1[i] = (vec2[i] > c) ? c : vec2[i];

  return(vec1);
}


/****************************************************************************/
/*   Subroutine vec_fill: fill m samples of vector with constant:            */
/*           vec[i] = c                                                      */
/****************************************************************************/
Float *vec_fill(Float *vec,Float c, int m)
{
   int i;
```

16:00:03

```
for(i=0; i < m; i++)
    vec[i] = c;

return(vec);
```

vect_fun.c.

02/03/99
16:00:16

fftreal.c.

```
********************** fftreal.c *******************

        Fast FFT of a real time sequence based on viewing
        the full-size real-data transform as an half-size
        complex-data transform.

        ifftr.c is the inverse of fftr.c  The two routines are
        complementary in the sense that the constants in w[](2)
        and br[] are the same and may be initialized only once
        by either routine.

        Y. Shoham  5/95        94/95 p. 178

     Modified by D. A. Kapilow 7/95, to speed it up on both the PC
     and SGI.


nc    "enhance.h"

****** Bit reversal function    *********/

     n        32-bit input integer
     ndim     Power-of-2 number. Log2(ndim) is the
              number of LS bits from "n" to reverse. The opreation is:

                     b[i] = b[ log2(nbit) -1 ] ,   i=0,log2(ndim) -1

              where b[i] is the bit at location i.

     The function returns an integer whose nbit LS-bits are the reversal
     of same bits in "n".  Other bits are 0.

tic int brvr(int n, int ndim)

     int j,m,k;

     m = 0;
     j = 1;
     for (k = ndim >> 1; k > 0; j <<= 1,k >>= 1)
             if (j & n)
                     m = m | k;
     return(m);


 Allocate and initialize the constant data for the FFT */
d fftrinit(
     Fftr     *fb,             /* out: initialize it */
     int      ln2size)         /* in: ln2(size) */

     int i, hsize, *br;
     Float t, pn, *w;

     fb->size = 1 << ln2size;
     hsize = fb->size >> 1;

     /* allocate coefficient and work arrays */
     if (((fb->cossin = (Float *)malloc(sizeof(Float) * fb->size)) == NULL ||
         ((fb->br = (int *)malloc(sizeof(int) * hsize)) == NULL)
                 fprintf(stderr, "malloc error\n");
     w = &fb->cossin[2];
     br = fb->br;
     pn = 2. * PI / fb->size;
     for (i = 1; i < hsize; i++) {
             t = pn * i;
             w[0] = cos(t);
```

```
                     w[1] = -sin(t);
                     w += 2;
             }
             for(i = 0; i < hsize; i++)
                     br[i] = brvr(i, hsize) << 1;
             fb->invsize = 1. / fb->size;
}

/* Free the cached data */
void fftrdone(Fftr *fb)
{
             free(fb->cossin);
             free(fb->br);
             fb->cossin = 0;
             fb->br = 0;
}

void fftr(
             Float    *y,      /* out: complex FFT */
             Float    *x,      /* in: real signal */
             Fftr     *fb)     /* in: cached FFT parameters */
{
             int      i, j, k, l, winc, ks, nsiz, nsizh, nsizq;
             int      *br;
             Float    t0, t1, u0, u1, w0, w1, pn;
             Float    *y0p, *y1p, *wp, *w;

             /* Initialization */
             nsiz = fb->size;
             br = fb->br;
             w = fb->cossin;
             nsizh = nsiz >> 1;
             nsizq = nsiz >> 2;

             /* Make the full-size real input an half-size complex */
             y0p = y;
             for(k = 0; k < nsizh; k++) {
                     y1p = &x[br[k]];
                     y0p[0] = y1p[0];
                     y0p[1] = y1p[1];
                     y0p += 2;
             }

             /* Half-size complex FFT */
             /* 1st stage nsizh/2 simple butterflies */
             y0p = y;
             for(k = 0; k < nsizq; k++) {
                     t0 = y0p[0];
                     t1 = y0p[1];
                     u0 = y0p[2];
                     u1 = y0p[3];
                     y0p[0] = t0 + u0;
                     y0p[1] = t1 + u1;
                     y0p[2] = t0 - u0;
                     y0p[3] = t1 - u1;
                     y0p += 4;
             }
             /* Next stages */
             for (i = 2, winc = nsizh; i < nsizh; winc >>= 1) {
                     ks = i - 1;
                     i <<= 1;
                     l = i << 1;
                     for (j = 0; j < nsiz; j += l) {
                             y0p = &y[j];
                             y1p = y0p + i;
```

02/05/99
16:00:16

fftreal.c.

```
                wp = &w[winc];
                t0 = y0p[0];
                t1 = y0p[1];
                u0 = y1p[0];
                u1 = y1p[1];
                y0p[0] = t0 + u0;
                y0p[1] = t1 + u1;
                y1p[0] = t0 - u0;
                y1p[1] = t1 - u1;
                for (k = 0; k < ks; k++) {
                        y1p += 2;
                        y0p += 2;
                        u0 = wp[0] * y1p[0] - wp[1] * y1p[1];
                        u1 = wp[0] * y1p[1] + wp[1] * y1p[0];
                        t0 = y0p[0];
                        t1 = y0p[1];
                        y0p[0] = t0 + u0;
                        y0p[1] = t1 + u1;
                        y1p[0] = t0 - u0;
                        y1p[1] = t1 - u1;
                        wp += winc;
                }
        }
}
/* Convert y to the final spectrum */
/* For 0 , nsizh/2 , nsizh terms */
y[nsizh + 1] = -y[nsizh + 1];
y0p = y;
y1p = &y[nsiz];
t0 = y0p[0];
t1 = y0p[1];
y0p[0] = t0 + t1;
y0p[1] = 0.;
y1p[0] = t0 - t1;
y1p[1] = 0.;

/* Other terms */
wp = &w[2];
for(k = 1; k < nsizq; k++) {
        y0p += 2;
        y1p -= 2;
        w0 = y0p[0];
        w1 = y1p[0];
        t0 = w0 + w1;
        u0 = w1   w0;
        w0 = y0p[1];
        w1 = y1p[1];
        t1 = w0 - w1;
        u1 = w0 + w1;
        w0 = wp[0];
        w1 = wp[1];
        pn = u0;
        u0 = w0 * u1 - w1 * u0;
        u1 = w0 * pn + w1 * u1;
        y0p[0] = 0.5 * (t0 + u0);
        y1p[0] = 0.5 * (t0 - u0);
        y0p[1] = 0.5 * (t1 + u1);
        y1p[1] = 0.5 * (u1 - t1);
        wp += 2;
}

********************* ifftr.c *********************

        Fast inverse FFT of a real time sequence based on viewing
        the full-size real-data transform as an half-size
        complex-data transform.

        ifftr.c is the inverse of fftr.c   The two routines are
        complementary in the sense that the constants in w[][2]
        and br[] are the same and may be initialized only once
        by either routine.


        Y. Shoham  5/95      94/95 p. 182

*/
void ifftr(
        Float   *y,     /* out: real signal */
        Float   *x,     /* in: complex FFT */
        Fftr    *fb)    /* in: cached FFT parameters */
{
        int     i, j, k, l, winc, ks, nsiz, nsizh, nsizq;
        int     *br, *brpl;
        Float   t0, t1, u0, u1, w0, w1, pn;
        Float   *y0p, *y1p, *wp, *w, *y3p;

        /* Initialization */
        nsiz = fb->size;
        br = fb->br;
        w = fb->cossin;
        nsizh = nsiz >> 1;
        nsizq = nsiz >> 2;

        /*
         * Convert input FFT to a spectrum of a half-size complex decimated
         * time sequence
         *
         * DC and nsiz/4 terms (and bit reversal)
         */
        y[0] = x[0] + x[nsiz];
        y[1] = x[0] - x[nsiz];
        i = br[nsizq];
        y[i] = 2.*x[nsizh];
        y[i+1] = -2.*x[nsizh + 1];

        y0p = x;
        y1p = &x[nsiz];
        /* Other terms (and bit reversal) */
        brpl = &br[nsizh-1];
        wp = w;
        for(k = 1; k < nsizq; k++) {
                wp += 2;
                y0p += 2;
                y1p -= 2;
                t0 = y1p[0] + y0p[0];
                pn = y0p[0] - y1p[0];
                t1 = y0p[1] - y1p[1];
                u1 = y0p[1] + y1p[1];
                w0 = wp[0];
                w1 = wp[1];
                u0 = w0 * u1 - w1 * pn;
                u1 = w0 * pn + w1 * u1;
                y3p = &y[br[k]];
                y3p[0] = t0 - u0;
                y3p[1] = t1 + u1;
                y3p = &y[*brpl--];
                y3p[0] = t0 + u0;
                y3p[1] = u1 - t1;
```

02/05/99
16:00:16

fftreal.c.

3

```
/* Half-size inverse FFT */
/* Do nsizh/2 simple butterflies*/
y0p = y;
for(k = 0; k < nsizq; k++) {
        t0 = y0p[0];
        t1 = y0p[1];
        u0 = y0p[2];
        u1 = y0p[3];
        y0p[0] = t0 + u0;
        y0p[1] = t1 + u1;
        y0p[2] = t0 - u0;
        y0p[3] = t1 - u1;
        y0p += 4;
}

/* Next stages */
for (i = 2, winc = nsizh; i < nsizh; winc >>= 1) {
        ks = i - 1;
        i <<= 1;
        l = i << 1;
        for (j = 0; j < nsiz; j += l) {
                y0p = &y[j];
                y1p = y0p + i;
                wp = &w[winc];
                t0 = y0p[0];
                t1 = y0p[1];
                u0 = y1p[0];
                u1 = y1p[1];
                y0p[0] = t0 + u0;
                y0p[1] = t1 + u1;
                y1p[0] = t0 - u0;
                y1p[1] = t1 - u1;
                for (k = 0; k < ks; k++) {
                        y0p += 2;
                        y1p += 2;
                        u0 = wp[0] * y1p[0] + wp[1] * y1p[1];
                        u1 = wp[0] * y1p[1] - wp[1] * y1p[0];
                        t0 = y0p[0];
                        t1 = y0p[1];
                        y0p[0] = t0 + u0;
                        y0p[1] = t1 + u1;
                        y1p[0] = t0 - u0;
                        y1p[1] = t1 - u1;
                        wp += winc;
                }
        }
}

/* Scale it */
t0 = fb->invsize;
for (k = 0; k < nsiz; k++)
        y[k] *= t0;
```

02/05/99
16:00:25

fftreal.h.

```
********************** rfft.c *********************

        Fast FFT of a real time sequence based on viewing
        the full-size real-data transform as an half-size
        complex-data transform.

        T. Shoham  5/95        94/95 p. 178

     Modified by D. A. Kapilow 7/95, to speed it up on both the PC
     and SGI.


pedef struct fftrcache

     int     size;           /* size of the FFT - power of 2 */
     int     *br;            /* Bit reversal index array of size nsiz/2 */
     Float   *cossin;        /* Complex FFT constants of size nsiz/2 */
     Float   invsize;        /* 1. / size */
rt.

id fftr(Float*,Float*,Pftr*);
id ifftr(Float*,Float*,Pftr*);
id fftrinit(Pftr*,int);
id fftrdone(Pftr*);
```

02/05/99
16:00:54

globals.h.

```
indef __globals__
 fine  __globals__
  . --------------------------------------------------------------------
 globals.h - Compilation Switches and Constants

 Author: Rainer Martin, AT&T Labs-Research

 Last Update: $Id:$

   ...... ............................................................
 .


 clude <stdio.h>
 clude <stdlib.h>
 clude <math.h>

  ..   * precision: choice of USEDOUBLES or USEFLOATS */
 ·ii    USEDOUBLES

 define the type of noise estimator to be used.
 MINSTAT is the optimal smoothing minimum statistics estimator
 MALAH is David Malahs noise estimation method. This is not fully implemented. */
 fine MINSTAT   /* choice of MALAH or MINSTAT */

 define the file format for the enhanced speech: WRITEFLOAT writes the data
 in the Float format which might be actually double or float.
 WRITESHORT writes 16 Bit short data. */
 fine WRITESHORT   /* choice of WRITESHORT or WRITEFLOAT */

 ****************************  CONSTANTS ******************************/

 fine PI              (Float)3.14159265358979323846


 idef USEDOUBLES
 pedef double Float;
 ndif

 idef USEFLOATS
 pedef float Float;
 ndif

 pedef short Word16;
 pedef long Word32;

 ndif
```

c2/05/99
16:01:08

windows.h.

```
(Float) 0.72480566402730,
(Float) 0.73569836841300,
(Float) 0.74644909611489,
(Float) 0.75705137209661,
(Float) 0.76749880994355,
(Float) 0.77778511650980,
(Float) 0.78790409570892,
(Float) 0.79784965224622,
(Float) 0.80761579529031,
(Float) 0.81719664208182,
(Float) 0.82658642147689,
(Float) 0.83577947742351,
(Float) 0.84477027236853,
(Float) 0.85355339059327,
(Float) 0.86212354147573,
(Float) 0.87047556267748,
(Float) 0.87860442325324,
(Float) 0.88650522668137,
    oat) 0.89417321381330,
     at) 0.90160376574032,
(..oat) 0.90879240657579,
(Float) 0.91573480615127,
(Float) 0.92242678262405,
(Float) 0.92886430500014,
(Float) 0.93504349555436,
(Float) 0.94096063217418,
(Float) 0.94661215059776,
(Float) 0.95199464656172,
(Float) 0.95710487785177,
(Float) 0.96193976625564,
(Float) 0.96649639941737,
(Float) 0.97077203259151,
(Float) 0.97476409029652,
(Float) 0.97847016786610,
(Float) 0.98188803289772,
(Float) 0.98501562659727,
(Float) 0.98785106501926,
(Float) 0.99039264020162,
(Float) 0.99263882119447,
(Float) 0.99458825498239,
(Float) 0.99623976729936,
(Float) 0.99759236333610,
(Float) 0.99864522833935,
     at) 0.99939772810259,
     at) 0.99984940934810,
(r.oat) 1.00000000000000,
(Float) 0.99984940934810,
(Float) 0.99939772810259,
(Float) 0.99864522833935,
(Float) 0.99759236333610,
(Float) 0.99623976729936,
(Float) 0.99458825498239,
(Float) 0.99263882119447,
(Float) 0.99039264020162,
(Float) 0.98785106501926,
(Float) 0.98501562659727,
(Float) 0.98188803289772,
(Float) 0.97847016786610,
(Float) 0.97476409029652,
(Float) 0.97077203259151,
(Float) 0.96649639941737,
(Float) 0.96193976625564,
(Float) 0.95710487785177,
(Float) 0.95199464656172,
(Float) 0.94661215059776,
```

```
(Float) 0.94096063217418,
(Float) 0.93504349555436,
(Float) 0.92886430500014,
(Float) 0.92242678262405,
(Float) 0.91573480615127,
(Float) 0.90879240657579,
(Float) 0.90160376574032,
(Float) 0.89417321381330,
(Float) 0.88650522668137,
(Float) 0.87860442325324,
(Float) 0.87047556267748,
(Float) 0.86212354147573,
(Float) 0.85355339059327,
(Float) 0.84477027236853,
(Float) 0.83577947742351,
(Float) 0.82658642147689,
(Float) 0.81719664208182,
(Float) 0.80761579529031,
(Float) 0.79784965224622,
(Float) 0.78790409570892,
(Float) 0.77778511650980,
(Float) 0.76749880994355,
(Float) 0.75705137209661,
(Float) 0.74644909611489,
(Float) 0.73569836841300,
(Float) 0.72480566402730,
(Float) 0.71377754671514,
(Float) 0.70262065700250,
(Float) 0.69134171618255,
(Float) 0.67994751826749,
(Float) 0.66844492669611,
(Float) 0.65684087019945,
(Float) 0.64514233862723,
(Float) 0.63335637873745,
(Float) 0.62149008995163,
(Float) 0.60955062007843,
(Float) 0.59754516100806,
(Float) 0.58548094430015,
(Float) 0.57336523722760,
(Float) 0.56120533759961,
(Float) 0.54900857016470,
(Float) 0.53678228179983,
(Float) 0.52453383716371,
(Float) 0.51227061426146,
(Float) 0.50000000000000,
(Float) 0.48772938573854,
(Float) 0.47546616283629,
(Float) 0.46321771820017,
(Float) 0.45099142983522,
(Float) 0.43879466240039,
(Float) 0.42663476277232,
(Float) 0.41451905561985,
(Float) 0.40245483899194,
(Float) 0.39044937992157,
(Float) 0.37850991004837,
(Float) 0.36664362126255,
(Float) 0.35485766137277,
(Float) 0.34315912980055,
(Float) 0.33155507330389,
(Float) 0.32005248173251,
(Float) 0.30865828381745,
(Float) 0.29737934299750,
(Float) 0.28622245328486,
(Float) 0.27519433517270,
(Float) 0.26430163158700,
```

02/05/99
16:01:08

windows.h.

```
    (Float) 0.25355090380511,
    (Float) 0.24294862790339,
    (Float) 0.23350119005645,
    (Float) 0.22221480349020,
    (Float) 0.21309590429108,
    (Float) 0.20215034775378,
    (Float) 0.19230420470969,
    (Float) 0.18280335791818,
    (Float) 0.17341357853311,
    (Float) 0.16422052257649,
    (Float) 0.15532972763147,
    (Float) 0.14664660940673,
    (Float) 0.13787645853427,
    (Float) 0.12952443733252,
    (Float) 0.12139557674676,
    (Float) 0.11349477331863,
    (F'oat) 0.10582678618670,
         at) 0.09839623425960,
    ,    at) 0.09120739342421,
    (Float) 0.08426519384873,
    (Float) 0.07757321737515,
    (Float) 0.07113569499986,
    (Float) 0.06495650444564,
    (Float) 0.05903936782582,
    (Float) 0.05338784940224,
    (Float) 0.04800535343828,
    (Float) 0.04289512214823,
    (Float) 0.03806023374436,
    (Float) 0.03350360058263,
    (Float) 0.02922796740849,
    (Float) 0.02523590970348,
    (Float) 0.02152983213390,
    (Float) 0.01811196710228,
    (Float) 0.01498437340273,
    (Float) 0.01214893498074,
    (Float) 0.00960735979838,
    (Float) 0.00736117880553,
    (Float) 0.00541174501761,
    (Float) 0.00376023270064,
    (Float) 0.00240763666390,
    (Float) 0.00135477166065,
    (F  oat) 0.00060227189741,
         t) 0.00015059065190,
    (..oat) 0.0

tic Float sqrt_tukey[256] = {

    (Float) 0.02066690122755,
    (Float) 0.04132497424881,
    (Float) 0.06196539442859,
    (Float) 0.08257934547233,
    (Float) 0.10315802119236,
    (Float) 0.12369263126935,
    (Float) 0.14417440400735,
    (Float) 0.16459459028073,
    (Float) 0.18494446727156,
    (Float) 0.20521534219563,
    (Float) 0.22539855601581,
    (Float) 0.24548548714080,
    (Float) 0.26546735510807,
    (Float) 0.28533622424911,
    (Float) 0.30508300733555,
    (Float) 0.32469946920468,
    (Float) 0.34417723036264,
```

```
    (Float) 0.36350797056383,
    (Float) 0.38268343236509,
    (Float) 0.40169542465297,
    (Float) 0.42053582614271,
    (Float) 0.43919658884737,
    (Float) 0.45766974151568,
    (Float) 0.47594739303707,
    (Float) 0.49402173381250,
    (Float) 0.51188504908960,
    (Float) 0.52952970226071,
    (Float) 0.54694815812243,
    (Float) 0.56413297409525,
    (Float) 0.58107681540194,
    (Float) 0.59777243820324,
    (Float) 0.61421271268967,
    (Float) 0.63039061612796,
    (Float) 0.64629923786094,
    (Float) 0.66193178225957,
    (Float) 0.67728157162574,
    (Float) 0.69234204904483,
    (Float) 0.70710678118655,
    (Float) 0.72156946105306,
    (Float) 0.73572391067313,
    (Float) 0.74956400374113,
    (Float) 0.76308406819981,
    (Float) 0.77627808076576,
    (Float) 0.78914050939639,
    (Float) 0.80166503569749,
    (Float) 0.81384871727019,
    (Float) 0.82568394999656,
    (Float) 0.83716647826253,
    (Float) 0.84829139711757,
    (Float) 0.85905395436989,
    (Float) 0.86944955261637,
    (Float) 0.87947375120649,
    (Float) 0.88912226013919,
    (Float) 0.89839098189198,
    (Float) 0.90727593318156,
    (Float) 0.91577332665506,
    (Float) 0.92387953251129,
    (Float) 0.93159108805128,
    (Float) 0.93890469915743,
    (Float) 0.94581724170063,
    (Float) 0.95232576287481,
    (Float) 0.95842748245825,
    (Float) 0.96411979400121,
    (Float) 0.96940026593933,
    (Float) 0.97426664263229,
    (Float) 0.97871684532735,
    (Float) 0.98274897304736,
    (Float) 0.98636130340272,
    (Float) 0.98955229332720,
    (Float) 0.99232057973705,
    (Float) 0.99466498011324,
    (Float) 0.99658449300667,
    (Float) 0.99807829846587,
    (Float) 0.99914575830730,
    (Float) 0.99978641670793,
    (Float) 1.00000000000000,
    (Float) 1.00000000000000,
    (Float) 1.00000000000000,
    (Float) 1.00000000000000,
    (Float) 1.00000000000000,
    (Float) 1.00000000000000,
    (Float) 1.00000000000000,
```

02/05/99
16:01:08

windows.h.

```
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(' ~at) 1.00000000000000,
    at) 1.00000000000000,
(..oat) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
    at) 1.00000000000000,
    at) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
```

```
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 1.00000000000000,
(Float) 0.99978641678793,
(Float) 0.99914575838730,
(Float) 0.99807829846587,
(Float) 0.99658449300667,
(Float) 0.99466498011324,
(Float) 0.99232057973705,
(Float) 0.98955229332720,
(Float) 0.98636130340272,
(Float) 0.98274897304736,
(Float) 0.97871684532735,
(Float) 0.97426664263229,
(Float) 0.96940026593933,
(Float) 0.96411979400121,
(Float) 0.95842748245825,
(Float) 0.95233576287481,
(Float) 0.94581724170063,
(Float) 0.93890469915743,
(Float) 0.93159108805128,
(Float) 0.92387953251129,
(Float) 0.91577332665506,
(Float) 0.90727593318156,
(Float) 0.89839098109198,
(Float) 0.88912226813919,
(Float) 0.87947375120649,
(Float) 0.86944955261637,
(Float) 0.85905395436989,
(Float) 0.84829139711757,
(Float) 0.83716647826253,
(Float) 0.82568394999658,
(Float) 0.81384871727019,
(Float) 0.80166583569749,
(Float) 0.78914050939639,
```

02/06/99
16:01:08

windows.h.

5

```
    (Float) 0.77627800876576,
    (Float) 0.76308406019901,
    (Float) 0.74956408374813,
    (Float) 0.73572391067313,
    (Float) 0.72156946105306,
    (Float) 0.70710678118655,
    (Float) 0.69234204904483,
    (Float) 0.67728157162574,
    (Float) 0.66193178225957,
    (Float) 0.64629923786094,
    (Float) 0.63039061612796,
    (Float) 0.61421271268967,
    (Float) 0.59777242820324,
    (Float) 0.58107601540196,
    (Float) 0.56413297609525,
    (Float) 0.54694815812243,
    (Float) 0.52952970226071,
    '    at) 0.51188504908960,
         at) 0.49402173581250,
    (Float) 0.47594739303707,
    (Float) 0.45766974151360,
    (Float) 0.43919658804737,
    (Float) 0.42053582614271,
    (Float) 0.40169542465297,
    (Float) 0.38268343236509,
    (Float) 0.36350797056383,
    (Float) 0.34417723036264,
    (Float) 0.32469946920468,
    (Float) 0.30508300733555,
    (Float) 0.28533622426911,
    (Float) 0.26546755510807,
    (Float) 0.24548548714080,
    (Float) 0.22539855601581,
    (Float) 0.20521534219563,
    (Float) 0.18494446727156,
    (Float) 0.16459459028073,
    (Float) 0.14417660400735,
    (Float) 0.12369261126935,
    (Float) 0.10315802119236,
    (Float) 0.08257934547233,
    (Float) 0.06196539462859,
    (Float) 0.04132497424881,
         at) 0.02066690122755,
         at) 0

ndif
```

WHAT IS CLAIMED IS:

1. A method for enhancing a speech signal for use in speech coding, the speech signal representing background noise and periods of articulated speech, the speech signal being divided into a plurality of data frames, the method comprising the steps of:

applying a transform to the speech signal of a data frame to generate a plurality of sub-band speech signals;

making a determination whether the speech signal corresponding to the data frame represents articulated speech;

applying individual gain values to individual sub-band speech signals, wherein the lowest permissible gain value for a frame determined to represent articulated speech is lower than the lowest permissible gain value for a frame determined to represent background noise only; and

applying an inverse transform to the plurality of sub-band speech signals.

2. The method of claim 1 further comprising the step of determining the individual gain values and wherein the lowest permissible gain value is a function of a lowest permissible *a priori* signal to noise ratio.

3. A method for enhancing a signal for use in speech coding, the signal being divided into data frames and representing background noise information and periods of articulated speech information, the method comprising the steps of:

making a determination whether the signal of a data frame represents articulated speech information; and applying a gain value to the signal, wherein the lowest permissible gain value for a frame determined to represent articulated speech is lower than the lowest permissible gain value for a frame determined to represent background noise only.

4. The method of claim 3 further comprising the step of determining the gain value and wherein the lowest permissible gain value is a function of a lowest permissible *a priori* signal to noise ratio.

FIG. 1

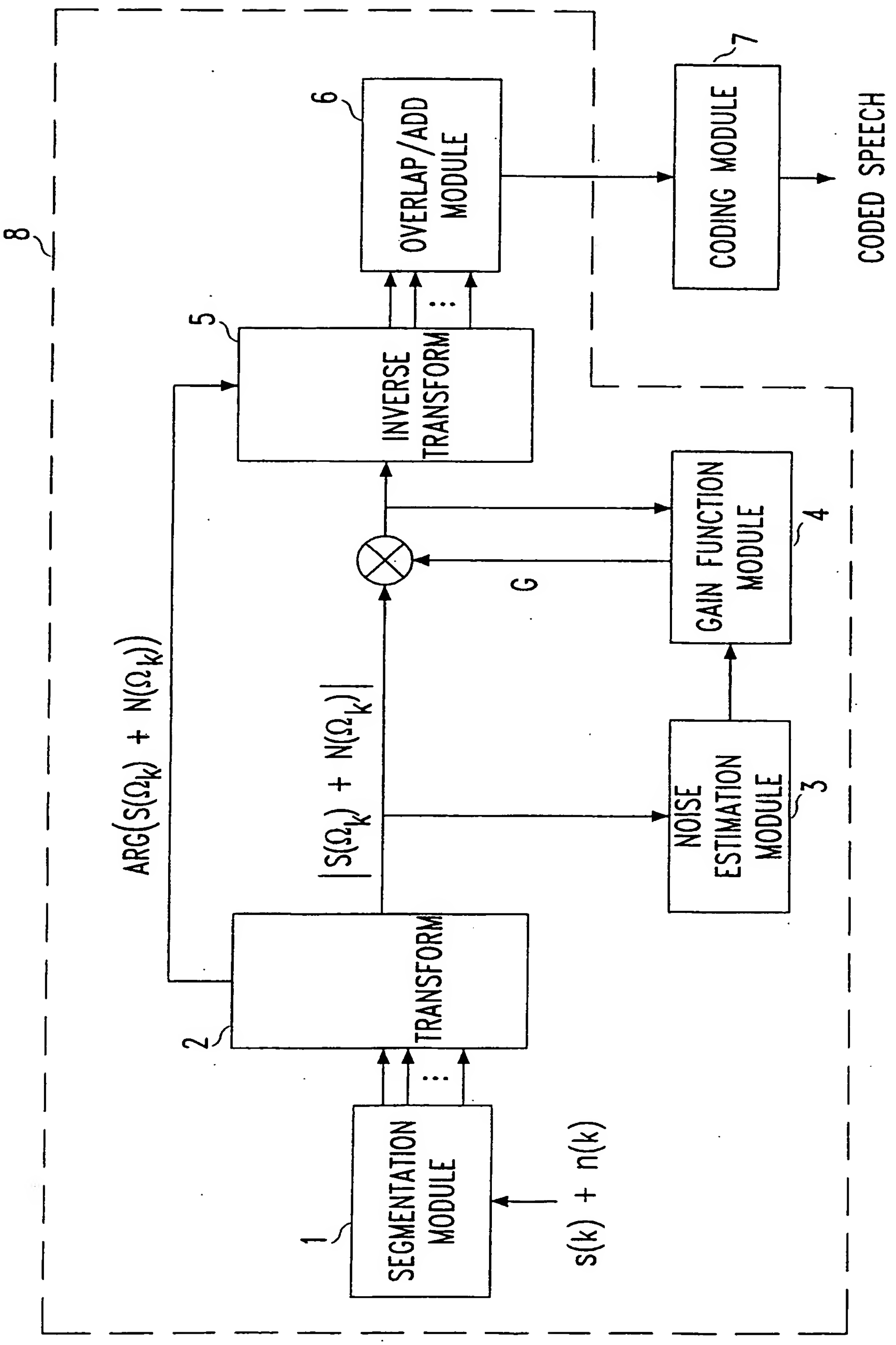

10
8
SEGMENTATION MODULE
1
$s(k) + n(k)$
TRANSFORM
2
$\text{ARG}(S(\Omega_k) + N(\Omega_k))$
$|S(\Omega_k) + N(\Omega_k)|$
NOISE ESTIMATION MODULE
3
GAIN FUNCTION MODULE
4
G
INVERSE TRANSFORM
5
OVERLAP/ADD MODULE
6
CODING MODULE
7
CODED SPEECH

FIG. 2

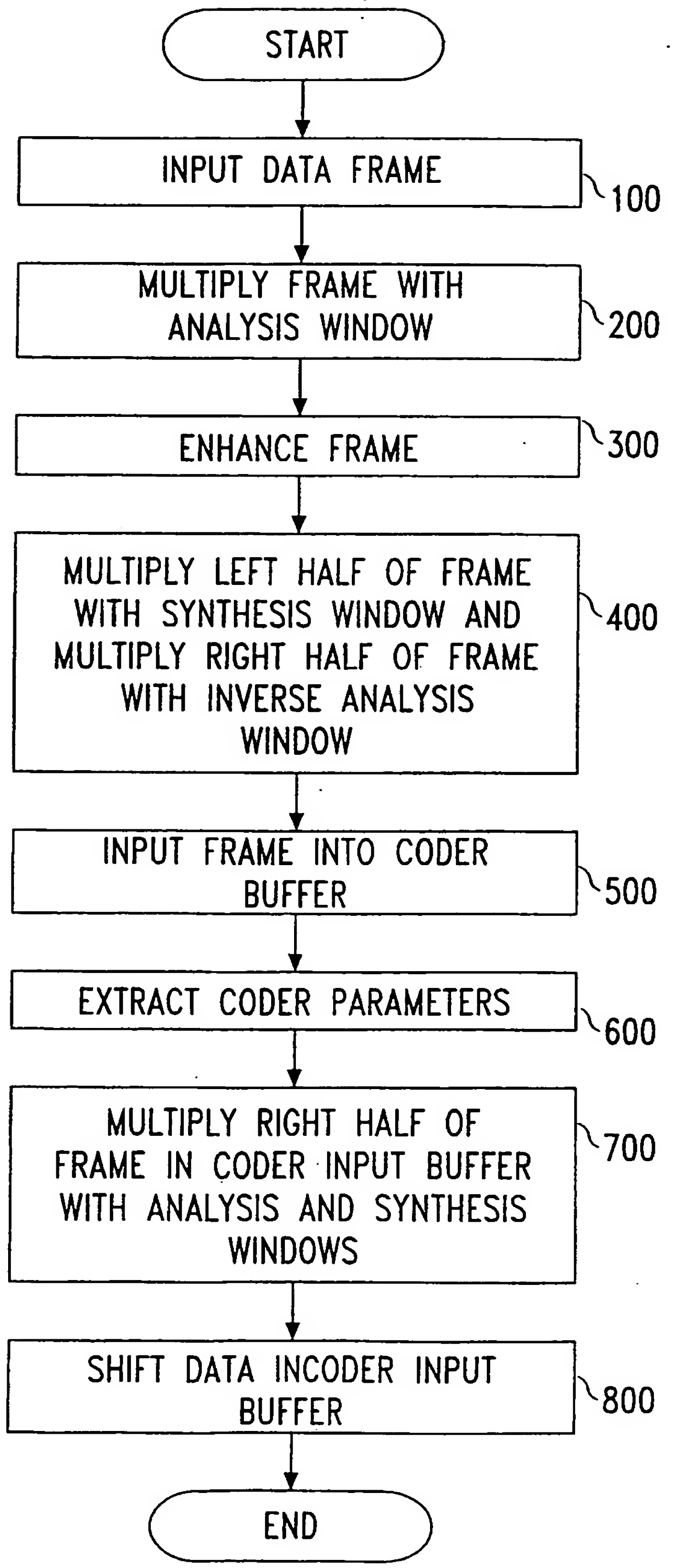

START
INPUT DATA FRAME
100
MULTIPLY FRAME WITH ANALYSIS WINDOW
200
ENHANCE FRAME
300
MULTIPLY LEFT HALF OF FRAME WITH SYNTHESIS WINDOW AND MULTIPLY RIGHT HALF OF FRAME WITH INVERSE ANALYSIS WINDOW
400
INPUT FRAME INTO CODER BUFFER
500
EXTRACT CODER PARAMETERS
600
MULTIPLY RIGHT HALF OF FRAME IN CODER INPUT BUFFER WITH ANALYSIS AND SYNTHESIS WINDOWS
700
SHIFT DATA INCODER INPUT BUFFER
800
END

FIG. 3

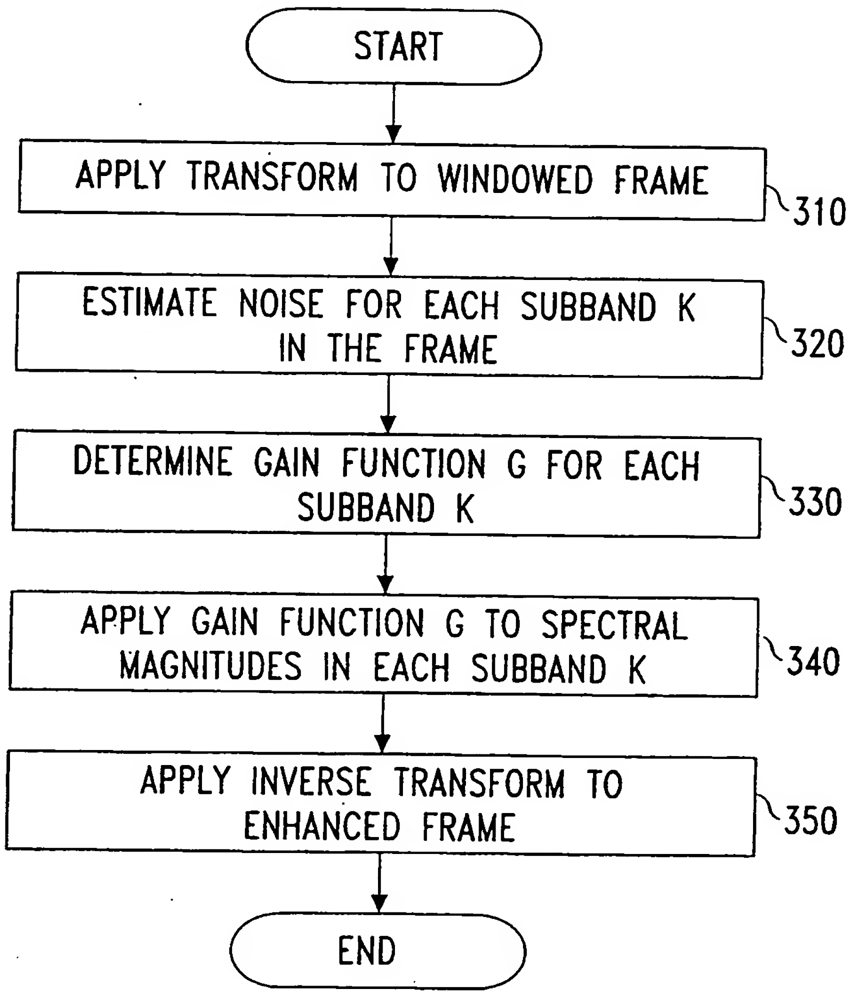

START
APPLY TRANSFORM TO WINDOWED FRAME
310
ESTIMATE NOISE FOR EACH SUBBAND K IN THE FRAME
320
DETERMINE GAIN FUNCTION G FOR EACH SUBBAND K
330
APPLY GAIN FUNCTION G TO SPECTRAL MAGNITUDES IN EACH SUBBAND K
340
APPLY INVERSE TRANSFORM TO ENHANCED FRAME
350
END

FIG. 4

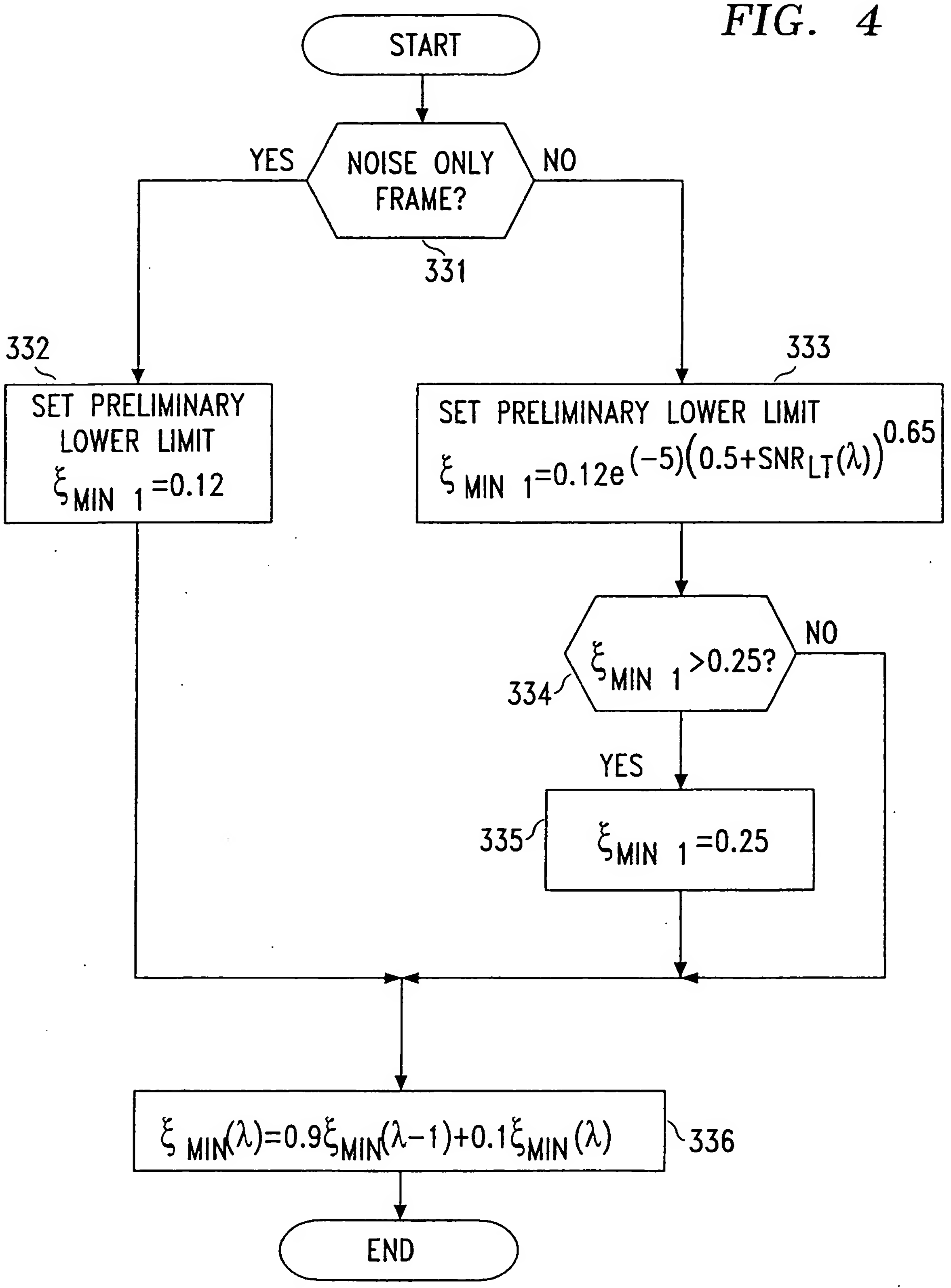
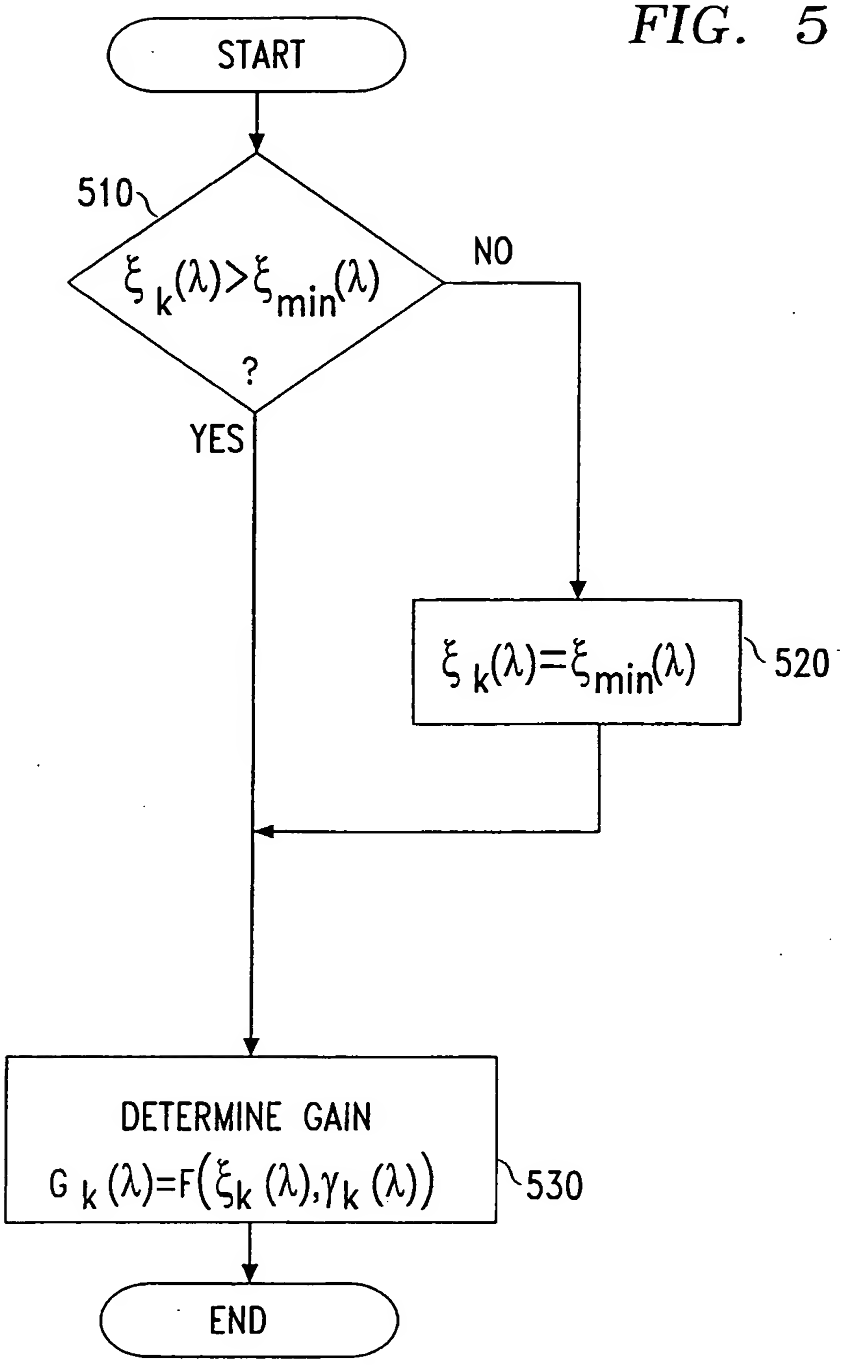

START
NOISE ONLY FRAME?
YES
NO
331
332
SET PRELIMINARY LOWER LIMIT $\xi_{MIN\ 1}=0.12$
333
SET PRELIMINARY LOWER LIMIT $\xi_{MIN\ 1}=0.12e^{(-5)(0.5+SNR_{LT}(\lambda))^{0.65}}$
334
$\xi_{MIN\ 1}>0.25?$
NO
YES
335
$\xi_{MIN\ 1}=0.25$
$\xi_{MIN}(\lambda)=0.9\xi_{MIN}(\lambda-1)+0.1\xi_{MIN}(\lambda)$
336
END

FIG. 5


START
510
$\xi_k(\lambda)>\xi_{min}(\lambda)$?
NO
YES
$\xi_k(\lambda)=\xi_{min}(\lambda)$
520
DETERMINE GAIN
$G_k(\lambda)=F(\xi_k(\lambda),\gamma_k(\lambda))$
530
END

INTERNATIONAL SEARCH REPORT

Inte :onal Application No
PCT/US 00/03372

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G10L21/02

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G10L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,X	MARTIN R ET AL: "New speech enhancement techniques for low bit rate speech coding" 1999 IEEE WORKSHOP ON SPEECH CODING PROCEEDINGS. MODEL, CODERS, AND ERROR CRITERIA (CAT. NO.99EX351), PROCEEDINGS OF 1999 IEEE WORKSHOP ON SPEECH CODING PROCEEDINGS. MODEL, CODERS, AND ERROR CRITERIA, PORVOO, FINLAND, 20-23 JUNE 1999, pages 165-167, XP002139862 1999, Piscataway, NJ, USA, IEEE, USA ISBN: 0-7803-5651-9 paragraph '0002! ---	1-4
A	US 5 839 101 A (HAEKKINEN JUHA ET AL) 17 November 1998 (1998-11-17) column 9, line 5 - line 45 column 10, line 24 - line 28 --- -/--	1,3

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search 9 June 2000	Date of mailing of the international search report 29/06/2000
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Krembel, L

1

INTERNATIONAL SEARCH REPORT

Intl. Application No
PCT/US 00/03372

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 012 519 A (AIZNER MENDEL ET AL) 30 April 1991 (1991-04-30) figure 13 column 10, line 18 - line 36	1,3
A	CAPPÉ O: "ELIMINATION OF THE MUSICAL NOISE PHENOMENON WITH THE EPHRAIM AND MALAH NOISE SUPPRESSOR" IEEE TRANSACTIONS ON SPEECH AND AUDIO PROCESSING,US,IEEE INC. NEW YORK, vol. 2, no. 2, 1 April 1994 (1994-04-01), pages 345-349, XP000575351 ISSN: 1063-6676 paragraph ‘0003!	1,3
A	SCALART P ET AL: "Speech enhancement based on a priori signal to noise estimation" 1996 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING CONFERENCE PROCEEDINGS (CAT. NO.96CH35903), 1996 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING CONFERENCE PROCEEDINGS, ATLANTA, GA, USA, 7-10 M, pages 629-632 vol. 2, XP002139863 1996, New York, NY, USA, IEEE, USA ISBN: 0-7803-3192-3 page 629, column 2, line 24 -page 630, column 1, line 11	1,3

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inte ional Application No
PCT/US 00/03372

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5839101 A	17-11-1998	FI 955947 A	13-06-1997
		AU 1067797 A	03-07-1997
		AU 1067897 A	03-07-1997
		EP 0790599 A	20-08-1997
		EP 0784311 A	16-07-1997
		WO 9722116 A	19-06-1997
		WO 9722117 A	19-06-1997
		JP 9212195 A	15-08-1997
		JP 9204196 A	05-08-1997
		US 5963901 A	05-10-1999
US 5012519 A	30-04-1991	NONE	